The VLDB Journal (2024) 33:1569–1589
https://doi.org/10.1007/s00778-024-00862-z

REGULAR PAPER

Flexible grouping of linear segments for highly accurate lossy compression of time series data

Xenophon Kitsios[1] · Panagiotis Liakos[1] · Katia Papakonstantinopoulou[1] · Yannis Kotidis[1]

Received: 4 November 2023 / Revised: 4 June 2024 / Accepted: 10 June 2024 / Published online: 15 July 2024

Abstract
Approximating a series of timestamped data points through a sequence of line segments with a maximum error guarantee is a fundamental data compression problem, termed as Piecewise Linear Approximation (PLA). As the demand for analyzing large volumes of time-series data across various domains continues to grow, the significance of this problem has recently received considerable attention. Recent PLA algorithms have emerged to help us handle the overwhelming amount of information, albeit at the expense of some precision loss. More precisely, these algorithms involve a delicate balance between the maximum acceptable precision loss and the space savings that can be achieved. In our recent work we proposed `Sim-Piece`, offering a fresh perspective on the long-standing challenge of PLO approximation. `Sim-Piece` identifies similarities among line segments in a PLA representation enabling their grouping and joint representation. This way, `Sim-Piece` delivers space-saving advantages that outperform even the optimal PLA approximation. In this work, we present `Mix-Piece`, an improved PLA compression algorithm that builds upon the core idea of `Sim-Piece` (i.e., exploiting similar PLA segments) but improves further its performance by (1) considering multiple candidate PLA segments when ingesting a time series, (2) enabling grouping of additional segments not utilized by `Sim-Piece`, and, (3) making use of a versatile output format that exploits all segment similarities. Our experimental evaluation demonstrates that `Mix-Piece` outperforms `Sim-Piece` and previous competing techniques, attaining compression ratios with more than twofold improvement on average over what PLA algorithms can offer. This allows for providing significantly higher accuracy with equivalent space requirements.

Keywords Compression · Lossy · Time series

1 Introduction

Contemporary applications generate massive amounts of streaming data, often depicted as a series of timestamped records. In many cases, storing the data is quite challenging due to their volume, and applying compression techniques as a means to reduce the respective storage requirements is deemed necessary. Depending on the problem at hand, one could use either lossless or lossy compression techniques. The former reduce the size of data without loss of information, whereas the latter aim for larger space savings while tolerating a bounded maximum error.

Piecewise linear approximation (PLA) is a fundamental data compression problem dating back to the 1960s [2], commonly used to approximate time series data. PLA algorithms represent time series measurements using a sequence of line segments, while keeping the approximation error within a predetermined acceptable threshold. Clearly, these algorithms are associated with a trade-off between space efficiency and precision loss, i.e., the space savings grow with the value of the error threshold.

PLA techniques have been shown to efficiently support the storage of voluminous historical time series data while also addressing many data analytics tasks, such as detecting seasonality and forecasting without sacrificing effectiveness [40]. Depending on the selected maximum error tolerance, PLA methods can drastically reduce the size of time series

✉ Panagiotis Liakos
panagiotisliakos@aueb.gr

Xenophon Kitsios
xkitsios@aueb.gr

Katia Papakonstantinopoulou
katia@aueb.gr

Yannis Kotidis
kotidis@aueb.gr

1 Athens University of Economics and Business, Athens, Greece

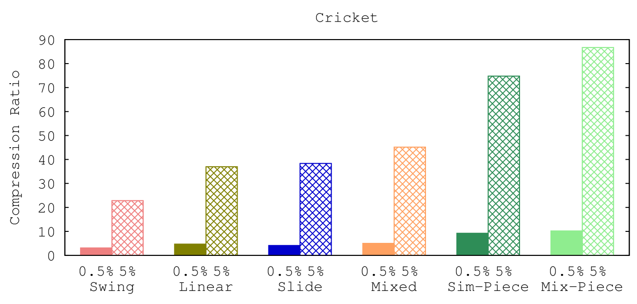

Fig. 1 Compression ratio comparison of previous PLA approaches for two error thresholds ϵ: a modest 5% and a strict one 0.5% of the dataset's range, against `Sim-Piece` and `Mix-Piece` algorithms

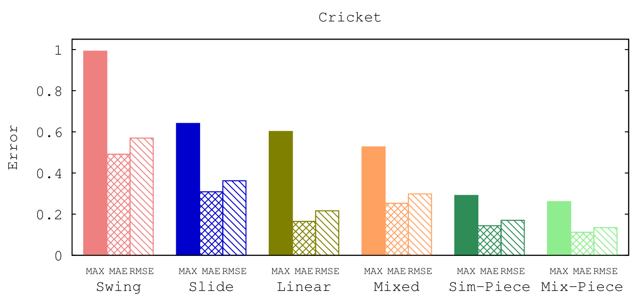

Fig. 2 Maximum Absolute Error (MAX), Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) comparison of previous PLA approaches for the same compression ratio (20:1), against `Sim-Piece` and `Mix-Piece` algorithms

data, achieving compression ratio values that are far beyond the capabilities of state-of-the art lossless compression algorithms [24].

1.1 Shortcomings of existing PLA techniques

In many applications there is a certain amount of imprecision in the collected data. For instance, sensor temperature measurements often have a 0.1–0.5 degree accuracy. For such applications, we may want to use a tight error threshold with respect to the range of values in order to reduce data size without sacrificing fidelity. Unfortunately, the merits of PLA approaches are not that evident in such a scenario and lossless [26] or bounded precision [28] compression techniques may provide larger space savings.

Figure 1 shows the compression ratio, defined as the proportion of the number of bytes in the uncompressed representation to the number of bytes in the compressed representation, achieved by earlier state-of-the-art algorithms that generate PLAs of a time series for a user-defined maximum error threshold.[1] The figure depicts results for a representative dataset from those used in our experimental evaluation and two error threshold values (ϵ), a modest one equal to 5%, and a strict one equal to 0.5%, of the signal's range (defined as the difference between its maximum and minimum value). With regards to the modest error threshold, we see that the performance of all four earlier PLA algorithms is sufficient. Slide [12] computes the optimal –space-wise– PLA representation [35] using disjoint line segments (discussed in Sect. 2) for a given maximum error threshold. Still, Mixed [29] offers increased space savings, by considering both joint and disjoint segments, taking up less than $1/45$ of the original size. However, when the error threshold is more stringent, the compression ratio of the earlier PLA approaches is less notable, reaching below 5:1 at its best (Mixed), whereas `Mix-Piece` achieves savings of 10:1.

[1] This figure is intended for illustration purposes only. A more thorough evaluation, including additional datasets and algorithms, can be found in our experimental evaluation section.

1.2 More efficient PLA via flexible grouping of segments

In this paper we discuss novel techniques that seek to exploit similarities among PLA-produced line segments in order to make PLA effective even on very tight error thresholds. This work builds upon our recent algorithm, `Sim-Piece`[24] that, to the best of our knowledge, was the first approach that exploited similar PLA segments' descriptions, in order to jointly represent them in the compressed output. As we see in Fig. 1, `Sim-Piece` and `Mix-Piece` greatly extend the space-savings of lossy approximations for the same error bounds. Moreover, our algorithms come up with *lossy* compressed representations that provide notable space savings, even in cases where the acceptable error threshold is very small.

When considering the quality of the approximation obtained by the different PLA techniques, Fig. 2 illustrates that, at an equivalent compression ratio (size), `Sim-Piece` and `Mix-Piece` bring remarkable enhancements in terms of the Maximum Absolute Error (MAX), i.e., the maximum absolute deviation of the approximate values against the input signal measurements. Moreover, our algorithms provide better approximation, when considering the Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) scores, surpassing even the performance of Linear [11], which utilizes a best-fit line per segment to improve approximation accuracy. This significant improvement can be attributed to the joint representation, which enables our approaches to represent a substantially greater number of similar PLA segments within the given space, as will be elaborated upon. This is evident in Fig. 3 which portrays the segments of `Mix-Piece` against the best earlier PLA algorithm (Mixed) when the respective representations occupy the same space. We see that `Mix-Piece` accommodates considerably more segments within the same space compared to Mixed, leading to a much more precise representation of the original values.

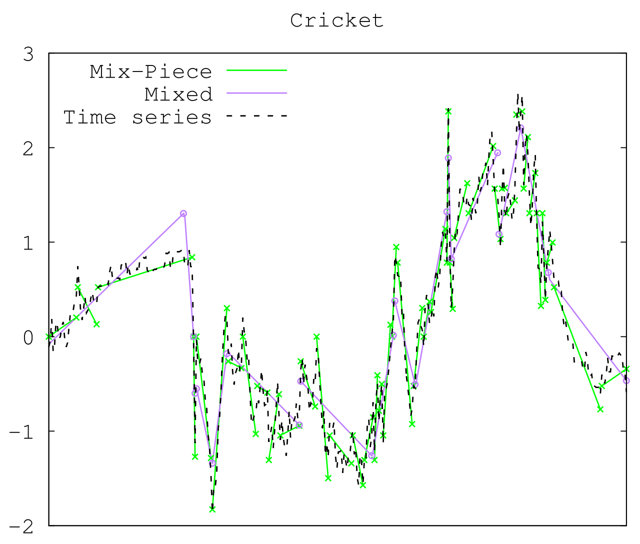

Fig. 3 Thanks to its flexible grouping, `Mix-Piece` employs a considerably greater number of similar segments within the same space compared to the runner-up PLA algorithm (Mixed), leading to a significantly more accurate representation of the original values.

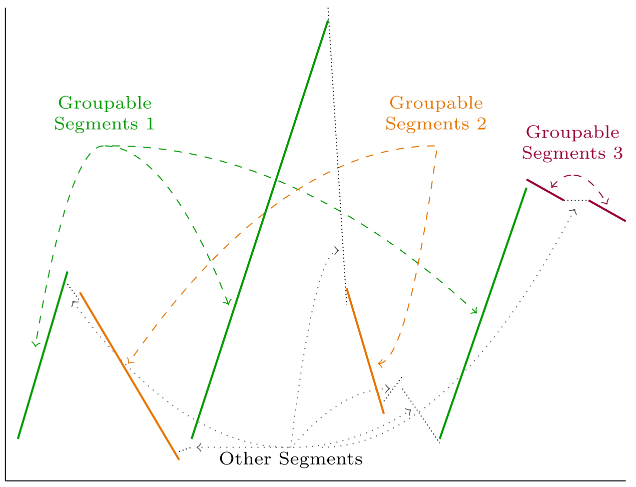

Fig. 4 We can group together the segments in green (Groupable Segments 1) and orange color (Groupable Segments 2) and represent them jointly with `Sim-Piece`. `Mix-Piece` may form additional groupings by considering segments even when they do not share a common starting point, as the case is with the two purple segments (Groupable Segments 3)

The operation of `Sim-Piece` is based on the following process. When considering a line segment, `Sim-Piece` fixes its starting point to a quantized value that is within ϵ of the original value. Then, we add subsequent data points to the segment by maintaining a pair of slopes, i.e., the extreme upper and lower slopes that satisfy the required approximation guarantees, until a point falls out of the area between them. As any of the lines between the two slopes can approximate the data points of the segment, we can find groups of segments with intersecting sets of candidate lines and represent them jointly. The latter process is illustrated in Fig. 4. The three segments in green color (Groupable Segments 1) and the two segments in orange (Groupable Segments 2) color have common starting points and similar slopes. Therefore, we can group them to reduce the overall space requirements of our representation. `Sim-Piece` computes the optimal solution to this problem, coming up with the minimum possible number of groups to induce impressive compression ratios.

Figure 4 also portrays two segments in purple color (Groupable Segments 3), that have a similar slope but do not share the same starting point. Consequently, if we grouped only those segments that have common starting points, as we do with `Sim-Piece`, the two purple (Groupable Segments 3) segments of Fig. 4 would remain *ungrouped*. Depending on the properties of the time series, such instances of segments that share comparable slopes but have distinct initial points could potentially present opportunities for achieving even greater space efficiency, which is not harnessed by `Sim-Piece`.

To emphasize this aspect, Fig. 5 features two real datasets, which will be further discussed in our experimental evaluation. The starting point distribution of segments of the time series of Fig. 5a is shown in b. As we can see, there are plenty of segments per starting point for a considerable number of groupings to be formed. However, this is not the case with the time series illustrated in Fig. 5c. This time series includes *trend*, and as we can see in Fig. 5d very few segments share a starting point. Thus, it is obvious that forming groups with `Sim-Piece` becomes more difficult when dealing with time series that include trend.

To address this as well as several additional challenges, we propose here a variation of `Sim-Piece`, termed `Mix-Piece`, that offers various performance improvements. `Mix-Piece` employs a more versatile output format than `Sim-Piece`, for better handling of time series that include *trend*. This format allows for exploiting similarities of line segments even when they do not share the same starting point value and significantly enhances the induced space savings. Additionally, `Mix-Piece` considers multiple candidate PLA segments when constructing a line segment, and ultimately uses the one that produces the longest segment. In this way, `Mix-Piece` generates fewer segments than Sim-Piece, and attains an increased compression ratio. This results in an even more accurate representation of the time series, as is shown in Figs. 2 and 3, compared to the state-of-the-art in PLA compression.

1.3 Summary of main findings

The performance gains our techniques achieve over a set of several real-world datasets are evident, regardless of whether we seek to maximize the space savings or to achieve high

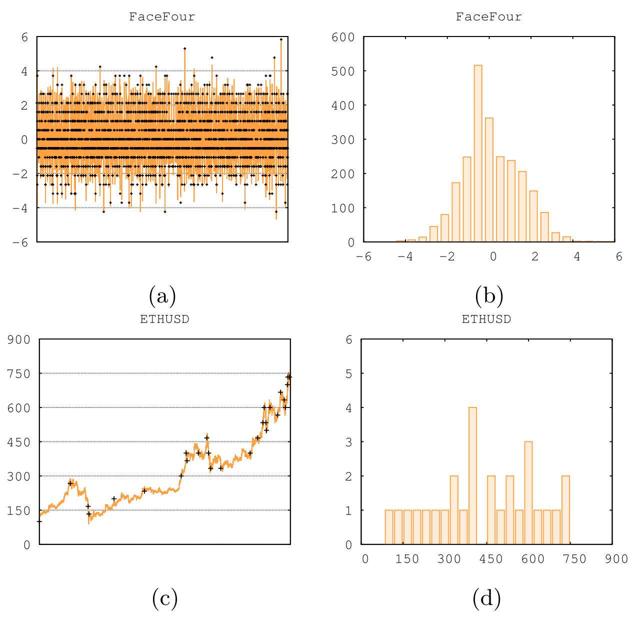

Fig. 5 Two time series, without (**a**) and with (**c**) trend, along with their respective starting point distributions of line segments (**b** and **d**)

accuracy by setting a large or small value of ϵ, respectively. More specifically, we show that our improvement over the best known PLA approaches in terms of space requirements [12, 29, 35], is consistent as ϵ grows, for values of ϵ that reach 30% of the dataset's range of values. In cases where accuracy is more important, we show that for the same compression ratio our algorithms produce approximations that are significantly more accurate for popular error metrics such as MAE and RMSE. Moreover, we show that our novel techniques outperform earlier PLA algorithms even when we use general purpose compression algorithms to reduce their requirements. On top of that, both `Sim-Piece` and `Mix-Piece` may also benefit from the use of such compression algorithms that help us reach even greater compression ratio. Comparing the new `Mix-Piece` algorithm against `Sim-Piece`, our results demonstrate that it offers between 3% and 74% relative improvement on the compression ratio it achieves.

Finally, we investigate the execution time of `Mix-Piece` and show that, despite its clear advantages in compression ratio, it is virtually as fast as `Sim-Piece`. Compared against previous approaches, `Mix-Piece` is on average 35× and 4× faster than the second and third best approaches with regard to space efficiency, respectively. In fact, the running time of `Mix-Piece` is comparable to that of Swing algorithm [12], which however offers very modest space savings.

We summarize here the key contributions of our proposed compression algorithm. In particular, we:

- propose `Mix-Piece`, an extension of the `Sim-Piece` lossy compression algorithm that significantly reduces the space requirements of PLA by identifying similar segments and merging their descriptions. `Mix-Piece` consistently outperforms `Sim-Piece` due to the more flexible grouping of a larger set of candidate PLA segments it takes into consideration. Our space gains allow for significantly increasing the accuracy we can obtain for the same compression ratio.
- show that the problem of grouping a set of candidate segments is solved optimally, as `Mix-Piece` always identifies the minimum number of groups required to collectively represent them in the algorithm's output. Moreover, we show that for the same PLA segmentation, the output of `Mix-Piece` is probably at least as small as that of `Sim-Piece`.
- demonstrate that by operating on independent groups of PLA segments, `Mix-Piece` is very efficient in terms of its running time while achieving better space-efficiency than what earlier approaches can offer.

2 Preliminaries

Let us now outline the background that will be needed in order to follow and understand our approach.

2.1 PLA methods with error threshold ϵ

PLA techniques read as input a potentially infinite stream of timestamped values $\langle t_i, v_i \rangle_{i \geq 0}$ and generate a number of line segments. For the setting we consider, these lines approximate the original values within a maximum error tolerance ϵ selected by the application.

Key to PLA segmentation is to derive the location and type of the knots [36, 38]. There are methods that enforce continuity conditions [36] at the knots (joint knots [12, 17, 20]), methods that allow discontinuity [36] (disjoint knots [12, 35, 38]) and techniques that actually consider both [29].

As mentioned in [36] the continuity requirement at the knots increases the complexity of the problem. Our segment matching algorithm can be adapted to work upon the output of an existing PLA segmentation (with joint or disjoint knots) by (1) adjusting the value at the starting knot, (2) reading the existing values and calculating upper and lower slopes of admissible approximation lines, for a given maximum error threshold, and (3) in the case of joint knots, trimming the last point of the segment, making it essentially disjoint [36].

For ease of exposition in the next section, we initiate the discussion of our algorithm via a pre-processing step (`Sim-Piece`$_{phase1}$) using a greedy PLA algorithm, which is an adaptation of Swing [12] that uses disjoint instead of joint knots. Our segment matching algorithm operates

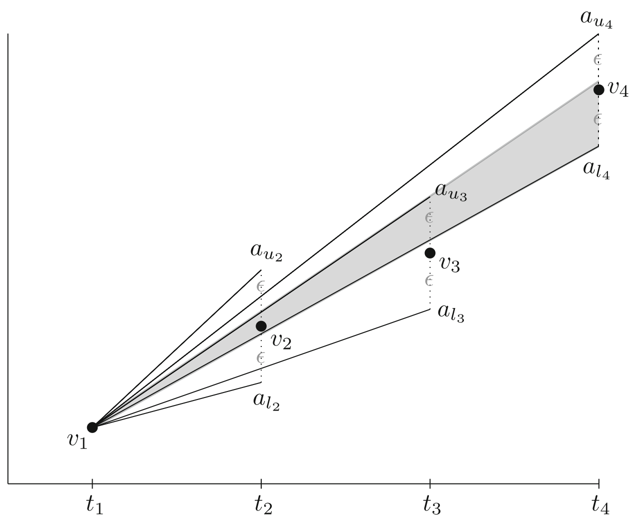

Fig. 6 Angle-based PLA. The gray area denotes all possible line segments starting from $\langle t_1, v_1\rangle$ that approximate the original values v_i within ϵ

directly on the output of this greedy algorithm without further modifications. In our exposition, a segment is described by (1) the timestamp t_i and value v_i of its starting point, and (2) the slope a_i of its line.

2.2 Angle-based greedy PLA

A common approach for constructing an approximate segment with a maximum error guarantee is to calculate extreme slope lines while adding new points. These lines help evaluate whether a new point can be approximated by the current segment, or is a *break-up point* that will trigger the creation of a new segment.

Figure 6 illustrates a process that follows this approach to approximate signal $\langle\langle t_1, v_1\rangle, \langle t_2, v_2\rangle, \langle t_3, v_3\rangle, \langle t_4, v_4\rangle\rangle$ using a fixed origin. This origin is set to be $\langle t_1, v_1\rangle$, i.e., the first point of the signal. When adding $\langle t_2, v_2\rangle$, the process of Fig. 6 creates an *angle* formed by the bounding slope lines a_{u_2} and a_{l_2} connecting $\langle t_1, v_1\rangle$ with $\langle t_2, v_2+\epsilon\rangle$ and $\langle t_2, v_2-\epsilon\rangle$, respectively. This angle specifies all lines that may approximate the two points within error threshold ϵ. The next point, i.e., $\langle t_3, v_3\rangle$ is more than ϵ away from both the upper and lower slopes. Therefore, when adding this point we need to reduce the angle so that the new slopes a_{u_3} and a_{l_3} pass through $\langle t_3, v_3+\epsilon\rangle$ and $\langle t_3, v_3-\epsilon\rangle$, respectively. Finally, adding $\langle t_4, v_4\rangle$ requires that the angle is further reduced, as the approximations provided by a_{l_3} are more than ϵ away from $\langle t_4, v_4\rangle$. Thus, a_{l_4} connecting $\langle t_1, v_1\rangle$ with $\langle t_4, v_4-\epsilon\rangle$ is set as the new lower slope. The upper slope is not updated, as the approximations provided by a_{u_3} are less than ϵ away from $\langle t_4, v_4\rangle$. The gray area of Fig. 6 bounded by a_{u_3} and a_{l_4} illustrates the final candidate lines that are less than ϵ away from all the points of the signal. If the distance of a point encountered was not within ϵ from either of the two slopes, it would trigger the creation of a new segment.

There has been extensive work following an approach such as the one described above, with the resulting segments being joint (Swing [12]), disjoint (Slide [12]) or both (Mixed [29]). In the case of disjoint knots, the break up point is set as the origin of a new segment, whereas in the case of joint knots the immediately previous point is used. Our suggested approach is an adaptation of Swing that produces disjoint knots with quantized associated values based on a maximum error threshold, and exploits the angle formed by the upper and lower slopes of each segment to represent them as intervals of slope values. As we will explain, these intervals will be used to *merge* the descriptions of different segments into a compact representation.

2.3 Interval graphs

The gray area formed by the angle of slopes a_{u_3} and a_{l_4} in Fig. 6 illustrates all the candidate lines that we may use to describe the respective segment within the defined accuracy. Thus, we can come up with an *interval* of slope values that captures all possible lines within the gray area. Naturally, the intervals of different segments will often intersect, enabling us to represent them *jointly*. We will next present the theorem that we will use to justify our method, along with the required definitions, to ease the understanding of our approach.

Definition 1 Consider a set of intervals $I = \{I_1, I_2, \ldots, I_n\}$ on a line for $n \in \mathbb{N}$, where $I_j = [a_{l_j}, a_{u_j}]$ and $a_{l_j} \leq a_{u_j}$, for $j \in \mathbb{N}$ and $j \leq n$. The interval graph $G = (V, E)$ formed by I contains one vertex v_j for each interval I_j, so $V = \{\mathsf{v}_1, \ldots, \mathsf{v}_n\}$, and there is an edge between vertices v_i and v_j if and only if I_i and I_j intersect, i.e., $E = \{(x_i, x_j) \mid I_i \cap I_j \neq \emptyset\}$.

Definition 2 A vertex is called *simplicial* if its neighbors form a clique, i.e., its neighbors are all linked to one another by edges.

Definition 3 A *perfect elimination scheme* in a graph with n vertices is an ordering $\mathsf{v}_1, \ldots, \mathsf{v}_n$ of the vertices such that v_i is simplicial in the graph that contains only vertices $\mathsf{v}_i, \ldots, \mathsf{v}_n$.

Interval graphs have the following characterization [16] that we are going to use later on to justify the optimality of our method.

Theorem 1 *A graph G is an interval graph, if, and only if, a perfect elimination scheme exists in it.*

3 Overview

We now discuss the details of our approach for high-precision storage reduction of time series data that is based on the idea that the different line segments produced by PLA techniques exhibit *similarities*. We first present a PLA technique that produces sets of candidate lines for each segment; we call these sets *intervals*. Then, we seek to *group* intervals that look alike in terms of their slope as well as their starting point, by identifying intersecting intervals. We provide algorithms for both these procedures which constitute the two phases of our `Sim-Piece` approach for highly accurate PLA with small storage footprint. Furthermore, we provide variations on both phases of `Sim-Piece`, to introduce our novel `Mix-Piece` algorithm, that addresses additional challenges and offers increased effectiveness.

3.1 Interval extraction

The first phase of `Sim-Piece` considers as input a data signal in the form of a sequence of discrete data points $\langle t_i, v_i \rangle$, where $i \in \{1, \ldots, n\}$, and an error threshold ϵ. The respective pseudocode is given with Algorithm 1.

We use the first point of the signal as the first point of the starting segment (Line 3). The number of discrete values that the starting points of our segments may take, can be arbitrarily large. However, we need to come up with a limited amount of *discrete* starting point values that multiple different segments may share, so that we are able to *jointly* represent them. In our context, we are interested in providing *approximations* within an error threshold ϵ. Therefore, we do not need to consider *all* the different original values. Instead, we can apply a quantization function such as the following for a value v, to come up with a quantized value b:

$$b = \lfloor v/\epsilon \rfloor \times \epsilon \tag{1}$$

As an example, values 1.1 and 1.4 would both result in $b = 1$ for $\epsilon = 0.5$. Using Eq. (1) we convert the first value v_s to the smallest multiple of ϵ that is within ϵ from the original value v_s (Line 4). We also initialize the upper (Line 5) and lower (Line 6) slopes forming the angle of the segment with the maximum and minimum possible values, respectively, so that we make sure the second point of the segment lies within the angle. Then we proceed by processing subsequent points in the signal, following the procedure discussed in Sect. 2.2 and depicted in Fig. 6. More specifically, we examine whether the distance of each point encountered is *not* within ϵ from the angle formed by the bounding lines with slopes a_u and a_l (Line 9). If so, we terminate the formation of the current segment by producing an interval for starting point b denoted by the final lower and upper slopes a_l and a_u and the initial timestamp t_s (Line 10), and we repeat the process considering the newly encountered point, i.e., $\langle t_c, v_c \rangle$, as the starting point of the next segment. Otherwise, depending on the position of the newly encountered point, we may need to update the angle by lowering the upper bounding slope or increasing the lower bounding slope. We achieve this by properly adjusting values a_u (Lines 15- 16) and a_l (Lines 17- 18), respectively.

Algorithm 1: `Sim-Piece`$_{phase1}$

```
Input: A data signal $s$: $\langle t_i, v_i \rangle$ $\forall i \in \{1, \ldots, n\}$, and an error threshold $\epsilon$
Output: An associative array $b\_intervals$, mapping each quantized $b$ value
        to a list of tuples $\langle a_l, a_u, t \rangle$
1  Function Sim-Piece$_{phase1}(s, \epsilon)$
2      $b\_intervals \leftarrow \{\{\}, \ldots, \{\}\}$;
3      $\langle t_s, v_s \rangle \leftarrow s$.next();
4      $b \leftarrow \lfloor v_s/\epsilon \rfloor \epsilon$;
5      $a_u \leftarrow \infty$;
6      $a_l \leftarrow -\infty$;
7      while $s$.hasNext() do
8          $\langle t_c, v_c \rangle \leftarrow s$.next();
9          if $v_c > a_u(t_c - t_s) + b + \epsilon$ or $v_c < a_l(t_c - t_s) + b - \epsilon$ then
10             $b\_intervals[b]$.add($\langle a_l, a_u, t_s \rangle$);
11             $\langle t_s, v_s \rangle \leftarrow \langle t_c, v_c \rangle$;
12             $b \leftarrow \lfloor v_s/\epsilon \rfloor \epsilon$;
13             $a_u \leftarrow \infty$;
14             $a_l \leftarrow -\infty$;
15         if $v_c < a_u(t_c - t_s) + b - \epsilon$ then
16             $a_u \leftarrow \frac{v_c + \epsilon - b}{t_c - t_s}$;
17         if $v_c > a_l(t_c - t_s) + b + \epsilon$ then
18             $a_l \leftarrow \frac{v_c - \epsilon - b}{t_c - t_s}$;
19     $b\_intervals[b]$.add($\langle a_l, a_u, t_c \rangle$);
20     return $b\_intervals$;
```

Following this procedure for all points of the signal, we come up with a list of intervals associated with each quantized b value encountered in the signal. The elements of the final lists comprise the upper and lower slope of each interval, which form the angle of each respective segment as illustrated in Fig. 6, as well as the initial timestamp of each segment.

3.2 Grouping lists of intervals

Using the intervals extracted through Algorithm 1, we aim to find the optimal groups of intervals for each quantized b value, so that we can represent similar intervals jointly and induce space savings.

Figure 7a depicts the angles of five intervals that share a common starting point b. We observe that the angles formed by the intervals may overlap, and thus, there exist candidate lines that may represent more than one interval. The flexibility of using any of the candidate lines of each interval –as they all satisfy the required approximation guarantees– enables us to group different overlapping intervals to minimize the space requirements of their representation. More specifically, our goal is to come up with the minimum number of groups of intersecting intervals for each starting point b. A formal description of this problem follows.

Problem 1 Given a set of intervals $I = \{I_1, I_2, \ldots, I_n\}$ as described in Definition 1, where a_{l_j} and a_{u_j} denote the minimum and maximum slopes of interval $I_j = [a_{l_j}, a_{u_j}]$, respectively, partition I into disjoint sets, so that all intervals in each set intersect, and the number of partitions is minimized.

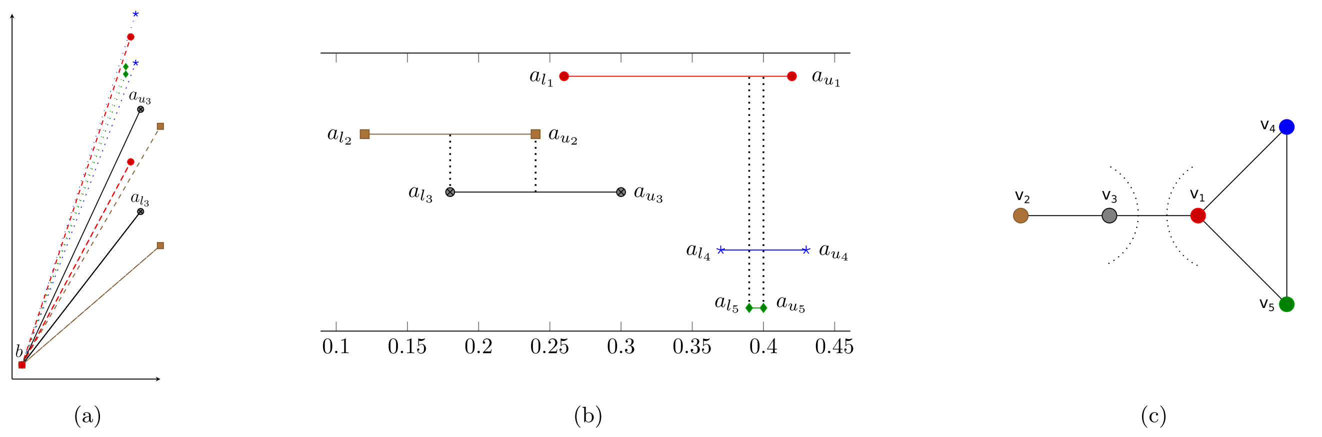

Fig. 7 Sets of candidate lines for 5 line segments, depicted by the maximum and minimum possible slope for each segment (**a**), the respective set of intervals (**b**), and the respective interval graph (**c**). There is no need to construct the interval graph, which is featured here only to ease understanding

In Fig. 7b we plot the sets of candidate lines of Fig. 7a as intervals, so that the overlapping areas are more evident. We observe that if we choose to group the interval $[a_{l_2}, a_{u_2}]$ with $[a_{l_3}, a_{u_3}]$, and the interval $[a_{l_1}, a_{u_1}]$ with $[a_{l_4}, a_{u_4}]$ and $[a_{l_5}, a_{u_5}]$ we will come up with a total of two groups. However, if we choose to group $[a_{l_1}, a_{u_1}]$ with $[a_{l_3}, a_{u_3}]$, we will come up with three groups in total, as we will form one with intervals $[a_{l_4}, a_{u_4}]$ and $[a_{l_5}, a_{u_5}]$ and one with a single interval $[a_{l_2}, a_{u_2}]$. That is, if we select to group $[a_{l_1}, a_{u_1}]$ with $[a_{l_3}, a_{u_3}]$, we forfeit the possibility of grouping all three $[a_{l_1}, a_{u_1}]$, $[a_{l_4}, a_{u_4}]$ and $[a_{l_5}, a_{u_5}]$ segments together, which forces us to come up with at least three groups, instead of just two, which is the optimal solution for this example.

The intersections among the different intervals can be represented even more clearly using the interval graph of Fig. 7c. An interval $[a_{l_i}, a_{u_i}]$ is represented by vertex v_i, and there exists an edge between two vertices v_i and v_j if the respective $[a_{l_i}, a_{u_i}]$ and $[a_{l_j}, a_{u_j}]$ intervals are overlapping. In this way, the problem of finding the minimum possible number of groups of overlapping intervals becomes equivalent with partitioning the vertices of the interval graph of Fig. 7c into the minimum number of groups such that the vertices in each group are all linked to one another by edges. This is a well studied problem, known as the 'minimum covering by disjoint completely connected sets or cliques' [19]. As the graph of Fig. 7c is an interval graph, there exists a perfect elimination scheme, which is very easy to determine. We simply need to choose a simplicial vertex, i.e., one whose neighbors are all linked to one another, and place it in the first position of our scheme. We then delete this vertex from the graph and look for a simplicial vertex in the remaining graph, which we place in the second position of our scheme and also delete from the graph. We continue doing this until the remaining graph is empty. Indeed, we can see that vertex v_2 is simplicial, as it has only one neighbor, so we can choose it for the first position of the elimination scheme. After deleting v_2,

Algorithm 2: `Sim-Piece`$_{phase2}$

```
Input: An associative array $b\_intervals$, mapping each quantized $b$ value to
       a list of tuples $\langle a_l, a_u, t\rangle$
Output: A list $groups$ containing one or more groups $\langle b, a_l, a_u, t\rangle$ for each
        quantized $b$ value
1  Function Sim-Piece$_{phase2}$ ($b\_intervals$)
2      $groups \leftarrow \{\}$;
3      for $intervals_{b_i} \in b\_intervals$ do
4          $group \leftarrow \langle b = b_i, a_l = -\infty, a_u = \infty, t = \{\}\rangle$;
           // sort by ascending $a_l$ order
5          sort($intervals_{b_i}$);
6          for $interval \in intervals_{b_i}$ do
7              if $interval.a_l \leq group.a_u$ and $interval.a_u \geq group.a_l$ then
8                  $group.a_u \leftarrow \min(group.a_u, interval.a_u)$;
9                  $group.a_l \leftarrow \max(group.a_l, interval.a_l)$;
10                 $group.t$.add($interval.t$);
11             else
12                 $groups$.add($group$);
13                 $group \leftarrow \langle b = b_i, a_l = interval.a_l, a_u = interval.a_u, t =$
                   $\{interval.t\}\rangle$;
14         $groups$.add($group$);
15     return $groups$;
```

we can choose v_3, which is also simplicial in the remaining graph, as it has only one neighbor left, i.e., v_1. Then, we can choose v_1, as its two remaining neighbors, i.e., v_4 and v_5 are all linked to one another. Next, we can choose v_4 and finally v_5. The final order of the perfect elimination scheme described above is:

$$\mathsf{v}_2 \rightarrow \mathsf{v}_3 \rightarrow \mathsf{v}_1 \rightarrow \mathsf{v}_4 \rightarrow \mathsf{v}_5$$

An optimal algorithm for coming up with a perfect elimination scheme is to sort the intervals in ascending order of the lower point of their interval, a_{l_i} [19]. As we can see in Fig. 7b, this order of the intervals matches the order of the perfect elimination scheme given above. Therefore, there is no need to construct the actual interval graph. We can simply follow the procedure described in Algorithm 2. We first initialize an empty list of groups to be produced (Line 2). Then, we iterate over the list of intervals of every b (Line 3), initialize the first group (Line 4) and order the respective intervals in ascending value of the lower point of their interval, a_{l_i} (Line 5). We go through the intervals in this order (Line 6) and consider them for placement in the current group. If the bounds of this group overlap with the bounds of the currently considered interval (Line 7), we add the interval to the group, by adjusting the bounds of the group accordingly and adding the timestamp of the interval to the group's list of timestamps (Lines 8–10). Otherwise, we close the group, placing it to the list of groups to be returned at the end of Algorithm 2 (Line 12), and place the interval in question in a new group (Line 13).

3.3 Mix-Piece

In this section, we propose enhancements on both phases of Sim-Piece, to introduce our novel Mix-Piece algorithm that provides significant improvements and addresses additional challenges.

During the creation of a new segment, Algorithm 1 applies Eq. 1 to the original value v_s, to come up with a quantized value b that is within ϵ from v_s. We argue here, that instead of using Eq. 1 we can also use Eq. 2:

$$b = \lceil v/\epsilon \rceil \times \epsilon \tag{2}$$

At the time of performing this quantization step, i.e., when we create a new segment, we are not aware of the value of the data points that follow and will be added to this segment. However, the choice of using Eq. 1 or Eq. 2 to quantize the starting point will have an impact on the length and quality of the resulting segment. Algorithm 3 considers both cases (Lines 4–5) and maintains two sets of intervals $[a_{l_1}, a_{u_1}]$ and $[a_{l_2}, a_{u_2}]$, corresponding to each of the two quantized values b_1 and b_2 as starting points, respectively. While processing the data points of the time series, we examine whether the current data point falls out of the area defined by each of the two intervals (Lines 15–18). We keep track of which interval comprises the most data points through variable $length$, that is updated depending on whether the data point fits in the area defined by the intervals (Lines 19–22). When a data point falls out of both the areas defined by the two intervals (Line 23), we use the one that comprises the most data points (Lines 24–27) and initiate the creation of a new segment (Lines 28–37). If the data point falls in the areas defined by the intervals, we update the two intervals accordingly (Lines 42–45).

Our experimental evaluation will reveal that this first enhancement introduced with Mix-Piece results in an improved compression ratio, as it reduces the number of segments that need to be grouped. Furthermore, we will demonstrate a significant enhancement in terms of Mean Absolute Error (MAE) and Root Mean Square Error (RMSE).

Algorithm 3: Mix-Piece$_{phase1}$

```
Input: A data signal $s$: $\langle t_i, v_i\rangle$ $\forall i \in \{1,\dots,n\}$, and an error threshold $\epsilon$
Output: An associative array $b\_intervals$, mapping each quantized $b$ value
        to a list of tuples $\langle a_l, a_u, t\rangle$
1  Function Mix-Piece$_{phase1}(s,\epsilon)$
2      $b\_intervals \leftarrow \{\{\},\dots,\{\}\}$;
3      $\langle t_s, v_s\rangle \leftarrow s$.next();
4      $b_1 \leftarrow \lfloor v_s/\epsilon \rceil \epsilon$;
5      $b_2 \leftarrow \lceil v_s/\epsilon \rceil \epsilon$;
6      $a_{u_1} \leftarrow \infty$;
7      $a_{l_1} \leftarrow -\infty$;
8      $a_{u_2} \leftarrow \infty$;
9      $a_{l_2} \leftarrow -\infty$;
10     $floor \leftarrow true$;
11     $ceil \leftarrow true$;
12     $length \leftarrow 0$;
13     while $s.hasNext()$ do
14         $\langle t_c, v_c\rangle \leftarrow s$.next();
15         if $v_c > a_{u_1}(t_c - t_s) + b_1 + \epsilon$ or $v_c < a_{l_1}(t_c - t_s) + b_1 - \epsilon$ then
16             $floor \leftarrow false$;
17         if $v_c > a_{u_2}(t_c - t_s) + b_2 + \epsilon$ or $v_c < a_{l_2}(t_c - t_s) + b_2 - \epsilon$ then
18             $ceil \leftarrow false$;
19         if $floor$ then
20             $length{+}{+}$;
21         if $ceil$ then
22             $length{-}{-}$;
23         if $!floor$ and $!ceil$ then
24             if $length > 0$ then
25                 $b\_intervals[b_1]$.add($\langle a_{l_1}, a_{u_1}, t_s\rangle$);
26             else
27                 $b\_intervals[b_2]$.add($\langle a_{l_2}, a_{u_2}, t_s\rangle$);
28             $\langle t_s, v_s\rangle \leftarrow \langle t_c, v_c\rangle$;
29             $b_1 \leftarrow \lfloor v_s/\epsilon \rceil \epsilon$;
30             $b_2 \leftarrow \lceil v_s/\epsilon \rceil \epsilon$;
31             $a_{u_1} \leftarrow \infty$;
32             $a_{l_1} \leftarrow -\infty$;
33             $a_{u_2} \leftarrow \infty$;
34             $a_{l_2} \leftarrow -\infty$;
35             $floor \leftarrow true$;
36             $ceil \leftarrow true$;
37             $length \leftarrow 0$;
38         if $v_c < a_{u_1}(t_c - t_s) + b_1 - \epsilon$ then
39             $a_{u_1} \leftarrow \frac{v_c+\epsilon-b_1}{t_c-t_s}$;
40         if $v_c > a_{l_1}(t_c - t_s) + b_1 + \epsilon$ then
41             $a_{l_1} \leftarrow \frac{v_c-\epsilon-b_1}{t_c-t_s}$;
42         if $v_c < a_{u_2}(t_c - t_s) + b_2 - \epsilon$ then
43             $a_{u_2} \leftarrow \frac{v_c+\epsilon-b_2}{t_c-t_s}$;
44         if $v_c > a_{l_2}(t_c - t_s) + b_2 + \epsilon$ then
45             $a_{l_2} \leftarrow \frac{v_c-\epsilon-b_2}{t_c-t_s}$;
46     if $length > 0$ then
47         $b\_intervals[b_1]$.add($\langle a_{l_1}, a_{u_1}, t_s\rangle$);
48     else
49         $b\_intervals[b_2]$.add($\langle a_{l_2}, a_{u_2}, t_s\rangle$);
50     return $b\_intervals$;
```

The second enhancement that Mix-Piece brings about, is focused on the grouping phase, and addresses challenges that Sim-Piece faces when dealing with time series that include *trend*. Algorithm 2 aims to form groups of segments that can be represented jointly. However, Sim-Piece considers two segments for grouping *only* if they share a common

Algorithm 4: `Mix-Piece`$_{phase2}$

```
Input: An associative array $b\_intervals$, mapping each quantized $b$ value to
    a list of tuples $\langle a_l, a_u, t\rangle$
Output: A list $groups_b$ containing one or more groups $\langle b, a_l, a_u, t\rangle$ for each
    quantized b value, a list $groups$ containing groups $\langle a_l, a_u, bt\rangle$ with
    grouped intervals not in $groups_b$, and a list $rest$ containing groups
    $\langle a_l, a_u, bt\rangle$ with ungrouped intervals.
1  Function Mix-Piece$_{phase2}$($b\_intervals$)
2      $groups_b \leftarrow \{\}$;
3      $ungrouped_b \leftarrow \{\}$;
4      for $intervals_{b_i} \in b\_intervals$ do
5          $group \leftarrow \langle b = b_i, a_l = -\infty, a_u = \infty, t = \{\}\rangle$;
           // sort by ascending $a_l$ order
6          sort($intervals_{b_i}$);
7          for $interval \in intervals_{b_i}$ do
8              if $interval.a_l \leq group.a_u$ and $interval.a_u \geq group.a_l$ then
9                  $group.a_u \leftarrow \min(group.a_u, interval.a_u)$;
10                 $group.a_l \leftarrow \max(group.a_l, interval.a_l)$;
11                 $group.t$.add($interval.t$);
12             else if $length(group) > 1$ then
13                 $groups_b$.add($group$);
14                 $group \leftarrow \langle b = b_i, a_l = interval.a_l, a_u = interval.a_u, t =$
                     $\{interval.t\}\rangle$;
15             else
16                 $ungrouped_b$.add($group$);
17                 $group \leftarrow \langle b = b_i, a_l = interval.a_l, a_u = interval.a_u, t =$
                     $\{interval.t\}\rangle$;
18         if $length(group) > 1$ then
19             $groups_b$.add($group$);
20         else
21             $ungrouped_b$.add($group$);
22     $groups \leftarrow \{\}$;
23     $rest \leftarrow \{\}$;
24     $group \leftarrow \langle a_l = -\infty, a_u = \infty, bt = \{\}\rangle$;
       // sort by ascending $a_l$ order
25     sort($ungrouped_b$);
26     for $interval \in ungrouped_b$ do
27         if $interval.a_l \leq group.a_u$ and $interval.a_u \geq group.a_l$ then
28             $group.a_u \leftarrow \min(group.a_u, interval.a_u)$;
29             $group.a_l \leftarrow \max(group.a_l, interval.a_l)$;
30             $group.bt$.add($\langle interval.b, interval.t\rangle$);
31         else if $length(group) > 1$ then
32             $groups$.add($group$);
33             $group \leftarrow \langle interval.a_l, a_u = interval.a_u, bt =$
                 $\{\langle interval.b, interval.t\rangle\}\rangle$;
34         else
35             $rest$.add($group$);
36             $group \leftarrow \langle a_l = interval.a_l, a_u = interval.a_u, bt =$
                 $\{\langle interval.b, interval.t\rangle\}\rangle$;
37     if $length(group) > 1$ then
38         $groups$.add($group$);
39     else
40         $rest$.add($group$);
41     return $\langle groups_b, groups, rest\rangle$;
```

starting point. This implies that there must be plenty of segments per starting point for a considerable number of groupings to be formed. This, however, is not the case with time series that include trend, such as the one of Fig. 5c. As we can see in the respective distribution of segments per starting point shown in Fig. 5d, very few segments do actually share a starting point.

`Mix-Piece` features a more versatile, bipartite grouping phase described with Algorithm 4. The first part (Lines 2–21) is very similar to what Algorithm 2 of `Sim-Piece` does.

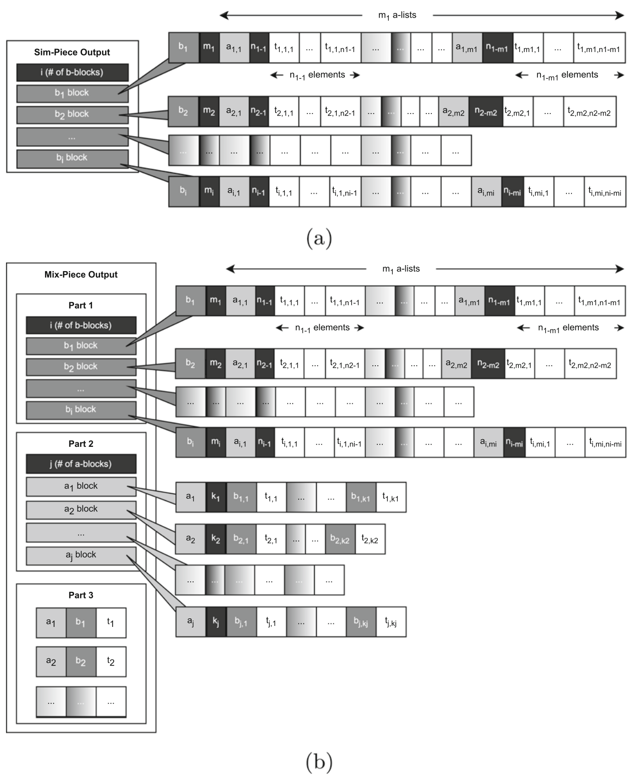

Fig. 8 The output of `Sim-Piece` (**a**) and `Mix-Piece` (**b**). We represent b values with dark gray, a values with light gray, timestamps with white, and lengths of sequences (blocks, a values, b values or timestamps) with black. The more versatile format of `Mix-Piece` comprises three parts in its output. The first part is identical with the output `Sim-Piece`, and the other two parts allow for increased savings

However, `Mix-Piece` gathers in set $ungrouped_b$ all intervals that were not grouped with any other interval sharing a common starting point, to be considered for grouping later on. Indeed, during the second part of Algorithm 4 (Lines 22–40) `Mix-Piece` attempts to group all remaining intervals, regardless of whether they share a common starting point. Of course, forming groups from these intervals necessitates storing their distinct starting values as well, resulting in slightly less gains compared to the formation of groups in the first part.

3.4 Output of `Sim-Piece` and `Mix-Piece`

The output of Algorithm 2 is a list of groups that comprise the quantized value of b, the upper and lower bounds a_u and a_l, and the starting-point timestamps of the grouped intervals. As we can use any of the lines within the area defined by the upper and lower bounds, we choose to use the line in the middle of this area, with $a = \frac{a_u + a_l}{2}$. This operation is applied after the grouping phase, and thus, it does not have any impact in the resulting compression ratio. The final output of `Sim-Piece` uses the compact representation shown in Fig. 8a, that does not waste space to repeat identical b or a values. It is evident that the proposed representation does not alter the original PLA segmentation, i.e., the position of knots, as denoted by the included timestamps. Moreover, by construction, the maximum error threshold guarantee still holds for the entire history of the time series.

The more versatile `Mix-Piece` comprises three parts in its output, that ultimately provide increased savings. More specifically, the first part of the output is *identical* with that of `Sim-Piece` and features intervals that can be added in groups of two or more intervals when groupings are formed between intervals *sharing a common starting point*. The second part represents intervals that are not included in the first part but can be added in groups of two or more intervals when we form groupings *regardless of their starting point*. Finally, the third part comprises those intervals that *do not overlap* with any interval and cannot be grouped. The final output of `Mix-Piece` is shown in Fig. 8b.

Every interval included in the second part of the output generated by `Mix-Piece` was initially grouped into a single-part group after the first grouping process of `Mix-Piece`, which groups only intervals that share a common starting point. These intervals are allocated to the second part of the output only if they are grouped with other intervals with which they overlap, enabling them to be jointly represented. Consequently, when dealing with the same set of intervals, the output size of `Mix-Piece` cannot exceed that of the `Sim-Piece` algorithm.

3.5 Optimality and complexity analysis

3.5.1 Analysis of `Sim-Piece`

The correctness and optimality of our segment grouping approach is guaranteed by the perfect elimination scheme property for interval graphs (Theorem 1). According to this property, which serves as the greedy choice of Algorithm 2, there is a certain order to examine the vertices, i.e., by ascending value of the lower point of their corresponding interval, so that each group formed contains the maximum possible number of vertices and no group is created unless it is really necessary. Therefore, we come up with the minimum possible number of intervals.

Algorithm 1 groups the n input time series values in batches of intervals, and then Algorithm 2 sort the batches of intervals that share the same starting point (quantized b value) and groups them. For each interval produced by Algorithm 1, we store exactly 4 elements: the a_{l_i} and a_{u_i} that define the slope of the interval, as well as the t_i and v_i that comprise the timestamped value. Hence the processing of each data point of the time series requires a constant amount of memory, and the number of intervals produced cannot exceed n, hence $O(n)$ space is needed. Moreover, as Algorithm 2 groups intervals, the space needed can only reduce, progressively. Therefore, the space complexity of our approach is $O(n)$.

Regarding the time complexity of the end-to-end process, Algorithm 1 is $O(n)$ time, since it processes the input values sequentially. Assume now that the number of intervals produced by Algorithm 1 is k. In the worst case with respect to running time, all intervals may be placed in the same batch, and the sorting step of Algorithm 2 will be computed in $O(k \log k)$ time. In practice, input segments are dispersed among multiple b values based on their starting points, resulting in even faster execution. The scanning and grouping of the sorted lists require $O(k)$ time.

3.5.2 Analysis of `Mix-Piece`

`Mix-Piece` differs from `Sim-Piece` in that i) it considers more PLA segments while processing the time series and ii) it performs additional groupings of intervals not considered by `Sim-Piece`.

Regarding the first aspect, Algorithm 3 as explained, proceeds by considering two segments, keeping in the end the one with the maximum number of points. While this necessitates additional computations of slopes, it does not affect the complexity of the process that is still $O(n)$.

Algorithm 4 performs grouping on intervals initially based on their starting point and, then for all that remain ungrouped after the first step. The whole process is still dominated by the sorting of the intervals in order to compute the perfect elimination scheme in each grouping stage. Thus, the complexity is again $O(klogk)$.

In practice, as our experimental evaluation will demonstrate, the initial stage of `Mix-Piece` is indeed slower than the corresponding phase in `Sim-Piece`, as anticipated. Nevertheless, due to Algorithm 3 generating fewer segments compared to Algorithm 1, the subsequent stage of `Mix-Piece` is often faster. This leads to both algorithms having comparable execution times.

As discussed in Sect. 3.4, for the same set of input intervals, the output of `Mix-Piece` cannot be larger than that of `Sim-Piece` due to the treatment of intervals that could not be grouped in the output of Algorithm 2.

Lemma 1 *For the same set of input PLA intervals, the compressed representation of the output of Algorithm 4 used by `Mix-Piece`, is always smaller than that of Algorithm 2 of `Sim-Piece`.*

Due to Algorithm 3 considering a greater number of candidate lines during PLA segmentation, the number of intervals forwarded to Algorithm 4 is reduced by 11% on average. Consequently, this leads to additional space savings in the output generated by `Mix-Piece`.

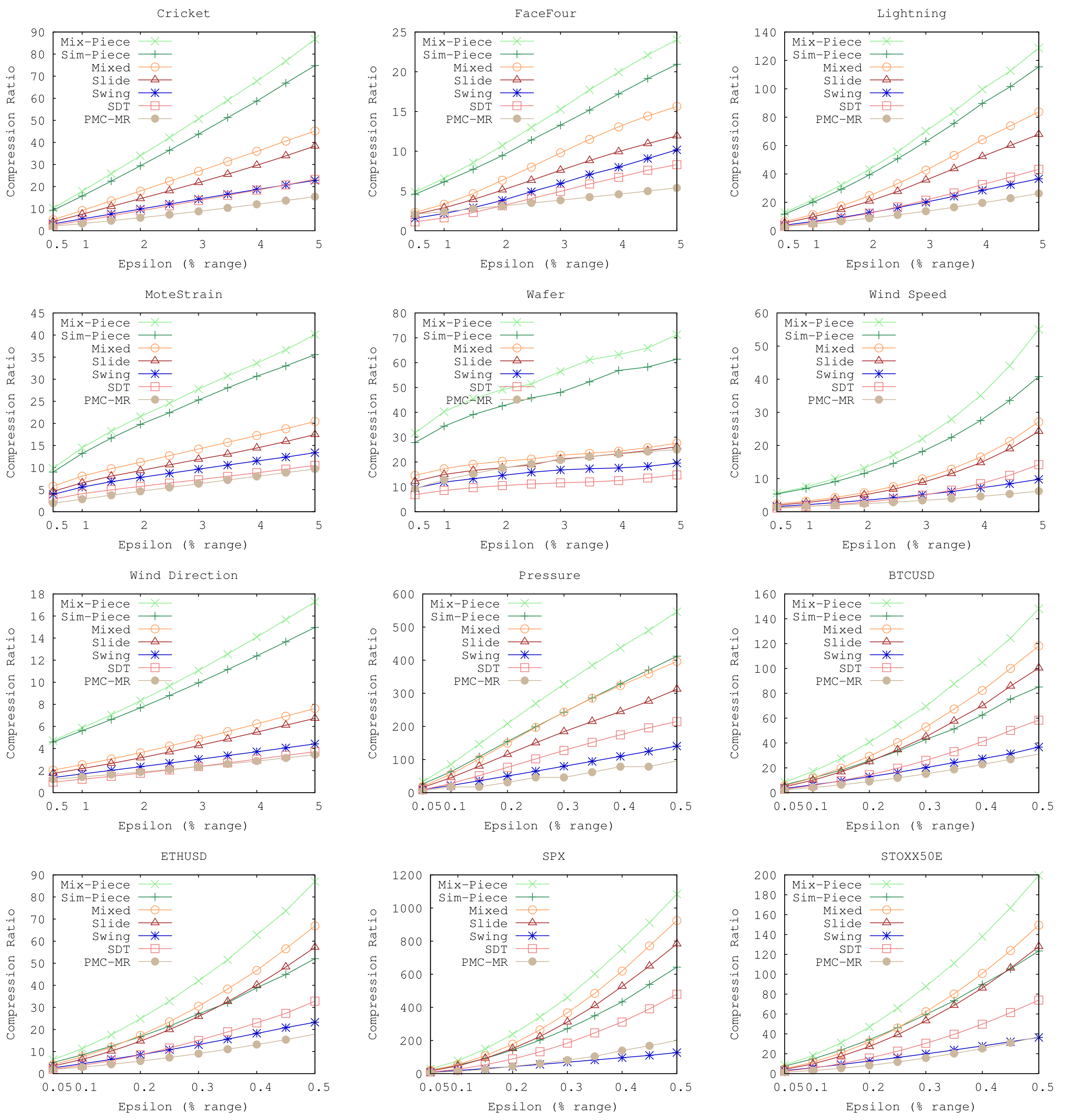

Fig. 9 Compression ratio results varying error threshold ϵ

4 Experimental results

We implemented our algorithms using Java and tested their performance against previous PLA algorithms. In this section we first present the dataset and technical details on our experiments. Then, we evaluate our algorithm by answering the following questions: i) What is the compression ratio that `Sim-Piece` and `Mix-Piece` achieve compared to earlier approaches as we vary the desired error threshold? ii) What is the level of accuracy our algorithms offer compared to other PLA techniques? iii) Can we induce further savings through general purpose compression? iv) How do the running times of `Sim-Piece` and `Mix-Piece` compare to those of other PLA algorithms?

Table 1 Details about our time series datasets, including the number of measurements (length), the original binary size, the minimum value, the number of decimal places, the range, the median value, and the standard deviation

Dataset	Length	Size (KB)	Min value	Decimal Places	Range (×0.5%)	Median	σ
Cricket	702,000	5484.38	−10.19918800	8	22.9 (0.115)	−0.041	1.0
FaceFour	39,200	306.25	−4.68758570	8	10.6 (0.053)	−0.098	1.0
Lightning	122,694	958.55	−1.78116300	8	24.9 (0.125)	−0.236	1.0
MoteStrain	106,848	834.75	−8.63799570	8	17.2 (0.086)	−0.003	1.0
Wafer	1,088,928	8507.25	−3.0539799	7	15.2 (0.076)	0.282	1.0
Wind speed	4,119,081	32,180.32	0.00	2	20.4 (0.102)	1.380	1.9
Wind dir	1,169,510	9136.8	0.00	2	360.0 (1.800)	186.850	107.2
Pressure	12,098,677	94,520.91	90.99386	5	13.1 (0.065)	101.125	3.2
BTCUSD	105,155	821.52	3882.22	2	25,404.1 (127,021)	9690.500	4240.2
ETHUSD	105,155	821.52	88.35	2	666.6 (3333)	243.570	142.9
SPX	1,807,286	14,119.42	1068.75	2	1629.3 (8.146)	1885.250	402.0
STOXX50E	1,122,336	8768.25	1924	0	1842.0 (9.210)	2988.000	419.3

4.1 Experimental setting

We ran our experiments on a computer with an Intel® Core™i7-7700 processor, with a 3.60GHz Base Frequency and a 4.20 GHz Max Turbo Frequency, a 8MB L3 cache, and a total of 16 GB DDR4 2400 MHz RAM. We implemented algorithms `Sim-Piece`, `Mix-Piece`, PMC-MR [25], SDT [1] and Swing [12] using Java and we used a C++ implementation for Slide [12], and Mixed [29] PLA algorithms, which the authors of [29] have generously provided. The output of all algorithms is in binary format and comprises for each segment an integer and a floating point value for PMC-MR and Swing, an integer and two floating point values for Slide and SDT, and an integer and either one or two floating point values for Mixed.

We note that based on the observations made by the authors of [29], Slide computes the minimum number of segments with disjoint knots for a maximum error bound and is thus optimal space-wise in this setting [35]. Moreover, Mixed has been shown [29] to provide even better savings, as it further considers joint knots.

Our dataset consists of 12 time series from four different sources. In all experiments we use synthetic timestamp sequences that start at 1 and increase by 1 at every data point. The properties of our dataset are listed in Table 1, and the different time series are thoroughly discussed below:

– UCR Time Series Classification Archive [3]

  - Cricket: Accelerometer data taken from actors performing cricket gestures.
  - FaceFour: Face outlines modeled as time series data.
  - Lightning: Transient electromagnetic events associated with lightning using a suite of optical and radio-frequency (RF) instruments.
  - MoteStrain: Humidity and temperature sensor data.
  - Wafer: A collection of inline process control measurements recorded from various sensors during the processing of silicon wafers for semiconductor fabrication.

– NEON datasets

  - WindSpeed [32]: Wind speed observations made by a wind monitor consisting of a propeller and vane design on lake and river buoys.
  - WindDirection [32]: Wind direction observations made by a wind monitor consisting of a propeller and vane design on lake and river buoys.
  - Pressure [31]: Barometric pressure on the meteorology station on the buoy in lakes and rivers.

– Cryptocurrencies' exchange rates [33]

  - BTCUSD: The USD exchange rate of Bitcoin in the year 2020, broken down into per minute values.
  - ETHUSD: The USD exchange rate of Ethereum in the year 2020, broken down into per minute values.

– Stock market indexes [33]

  - SPX: The Standard and Poor's 500 stock market index, representing 500 of the largest companies listed on US stock exchanges, for the years 2010–2017, presented as per-minute values.
  - STOXX50E: The EURO STOXX 50 stock market index, representing 50 major blue-chip companies

from the eurozone, for the years 2010-2017, presented as per-minute values.

4.2 Compression ratio comparison

We start our evaluation by measuring the space required to compress each of the 12 datasets. Figure 9 shows the compression ratio achieved by `Sim-Piece` and `Mix-Piece` compared to earlier algorithms. We report the compression ratio for varying error threshold ϵ values, expressed as a percentage of the total range of each dataset, listed in Table 1. More specifically, we report results for $0.5\% \times range \leq \epsilon \leq 5\% \times range$ for the first 7 datasets. However, for the remaining five datasets, we employed narrower error thresholds ($0.05\% \times range \leq \epsilon \leq 0.5\% \times range$). This adjustment was made to ensure that compression ratio values remained within comparable ranges, given that the latter datasets were inherently easier to compress using all PLA techniques.

We see in Fig. 9 that `Mix-Piece` clearly stands out as the most space-efficient algorithm. Moreover, the improved performance of `Mix-Piece` over earlier PLA approaches is evident for all values of ϵ investigated.

We further observe that in all figures `Mix-Piece` provides larger space savings than our earlier `Sim-Piece` algorithm. This is attributed to the following two alternative strategies of `Mix-Piece`: i) the use of two possible starting points for each segment, and ii) the more flexible grouping of segments performed by the algorithm. The first strategy of `Mix-Piece` creates a segment for each of the two possible starting points and ultimately uses the larger segment, which naturally results in an improved overall compression ratio. The second strategy of `Mix-Piece` attempts to form additional groupings compared to `Sim-Piece` which may lead to increased savings. The latter are most evident in cases of time series that include trend. This is clear in the results of Table 2 where we report the groupings performed after the execution of Algorithm 4. We observe that for the first 7 datasets most of the groupings occur between segments that share a common starting point. The respective reduction percentage is between 94.3% and 99.9%. On the contrary, the intervals of the generated line segments for the last 4 time series of our dataset that include trend, do not often overlap with the intervals of other segments with the same starting point. Thus, the respective reduction percentage is much smaller, ranging between 21.5% and 54.6% for $\epsilon = 0.5\%$ of the dataset's range. For these time series, we see that most of the groupings are performed during the second part of Algorithm 4 that groups segments regardless of their starting point. The extra groupings performed by `Mix-Piece`, as well as the improved representation for the segments that remain ungrouped, contribute to the increased compression ratio it offers. The output format that `Mix-Piece` uses for the different types of groupings guarantees that the size of the representation will not be larger than that of `Sim-Piece` regardless of the number of additional groupings achieved. However, the savings grow with this number. With regards to the Pressure dataset, the reduction range is 76% for $\epsilon = 0.5\%$ and 30.7% for $\epsilon = 5\%$ of the dataset's range, which translates to notable savings in terms of compression ratio. When setting $\epsilon = 5\%$ of the dataset's range for the last 4 datasets very few segments are generated. However, it is evident that most groupings again occur in the second part of Algorithm 4.

Table 3 presents the relative compression ratios of each algorithm compared to `Mix-Piece`, the top-performing algorithm, across different datasets. The values are calculated using the rightmost ϵ parameter value from the corresponding Fig. 9 graphs, and this parameter is explicitly mentioned alongside the dataset names for clarity. The rightmost column of the table displays the average relative performance over all datasets. The second-best algorithm, `Sim-Piece`, achieves an average compression ratio of 82%, signifying its commendable performance, as it is 18% below that of `Mix-Piece`. In comparison, Mixed and Slide attain relative compression ratios of 63% and 53%, respectively, compared to `Mix-Piece`. SDT and PMC-MR exhibit subpar performance across most datasets. As a result, these two algorithms were omitted from the rest of our experiments for brevity.

4.3 Quality of approximation

Both `Sim-Piece` and `Mix-Piece` are designed to offer maximum error guarantees to each and every value of the time series. Thus, it is worth exploring i) how the actual Mean Absolute Error (MAE) compares to the maximum error threshold used, and ii) what is the Root Mean Square Error (RMSE) of the approximation. For the latter we used an unmodified version for both algorithms, that does not seek to optimize this metric by adjusting the computation of the slope values accordingly.

In Table 4 we report results for all datasets and PLA techniques. For each measurement, we compute the average of datasets separately for ϵ=5% and ϵ=0.05%. Additionally, we evaluate the percentage deviation of these averages from Mix-Piece. We report MAE as a fraction of the dataset range (MAEr%), in order to have a comparison of the obtained approximation with respect to the maximum error threshold ϵ used that is also shown for each dataset. We notice that all methods provide approximations that are significantly more accurate than the requested threshold. In fact, the average MAEr% in all algorithms is about half of the ϵ value requested. Moreover, the reported RMSE values are very close to the measured MAE. This implies that there is a very small deviation among the errors obtained on the individual measurements. When compared to the other techniques, `Mix-Piece` consistently delivers the smallest MAE and RMSE scores, respectively. What's remarkable is that it

Table 2 Groupings performed during the execution of Algorithm 4 and respective reduction percentage

Dataset	0.5%				5%			
	# Of segments	Grouped (same starting point)	Rest grouped	Not grouped	# Of segments	Grouped (same starting point)	Rest grouped	Not grouped
Cricket	124,217	123,347 (99.3%)	818	52	13,993	13,914 (99.4%)	23	56
FaceFour	13,178	12,422 (94.3%)	743	13	2423	2298 (94.8%)	80	45
Lightning	16,271	15,780 (97.0%)	429	62	1385	1274 (92.0%)	53	58
MoteStrain	16,507	15,591 (94.5%)	850	66	4534	4482 (98.9%)	18	34
Wafer	62,435	61,342 (98.2%)	1004	89	29,212	29,089 (99.6%)	57	66
Wind Speed	1,455,674	1,455,130 (99.9%)	528	16	143,747	143,694 (99.9%)	24	29
Wind Dir	459,697	457,320 (99.5%)	2324	53	128,454	128,441 (99.9%)	2	11
Pressure	29,664	22,555 (76.0%)	6756	353	1106	340 (30.7%)	402	364
BTCUSD	832	212 (25.5%)	589	31	14	0 (0.0%)	2	12
ETHUSD	1480	520 (35.1%)	942	18	28	2 (7.1%)	8	18
SPX	1891	407 (21.5%)	1362	122	20	0 (0.0%)	0	20
STOXX50E	7182	3920 (54.6%)	3214	48	100	4 (4.0%)	24	72

Table 3 Relative compression ratio compared to mix-piece

	Dataset												
Epsilon	Cricket 5%	FaceFour 5%	Lightning 5%	MoteStrain 5%	Wafer 5%	Wind speed 5%	Wind Dir. 5%	Pressure 0.5%	BTCUSD 0.5%	ETHUSD 0.5%	SPX 0.5%	STOXX50E 0.5%	Average
Mix-Piece	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Sim-Piece	0.88	0.91	0.90	0.91	0.95	0.81	0.90	0.82	0.67	0.67	0.72	0.73	0.82
Mixed	0.52	0.65	0.65	0.51	0.39	0.49	0.44	0.73	0.80	0.77	0.85	0.75	0.63
Slide	0.44	0.50	0.53	0.44	0.37	0.44	0.39	0.57	0.68	0.66	0.72	0.64	0.53
Swing	0.26	0.42	0.28	0.33	0.27	0.18	0.26	0.26	0.25	0.27	0.12	0.18	0.26
SDT	0.20	0.26	0.25	0.20	0.16	0.19	0.16	0.29	0.30	0.28	0.33	0.28	0.24
PMC-MR	0.18	0.22	0.20	0.24	0.35	0.11	0.20	0.18	0.21	0.21	0.18	0.19	0.21

achieves this high level of accuracy while using considerably less space, as depicted in Fig. 9, when compared to all other techniques. To provide a clearer perspective on the impressively improved accuracy of `Mix-Piece`, we present a plot in Fig. 10, which illustrates the MAE values obtained by each algorithm for the same approximation size, specifically for the Wafer dataset. It becomes evident that for equivalent output sizes (x-axis values), `Mix-Piece` achieves significantly lower MAE scores, resulting in PLA representations that are substantially more accurate. Fig. 10 also reports the performance of the Linear algorithm [11], that seeks to optimize the approximation by using a best-fit line within each segment. Impressively, both `Sim-Piece` and `Mix-Piece` surpass its performance. Similar patterns were observed for the other datasets, although those figures are omitted for brevity.

4.4 Impact of error threshold

We continue our investigation on the performance of `Sim-Piece` and `Mix-Piece`, by examining the impact of the error threshold ϵ on their compression ratio. For every approach we measure the compression ratio achieved as ϵ grows up to 50% of each dataset's range. Larger values of ϵ are not meaningful as they would enable us to represent the entire signal using a single constant value. The performance is similar for all datasets and thus, for brevity, we illustrate (in logarithmic scale) the results for only two of our datasets in Fig. 11.

We see that `Sim-Piece` and `Mix-Piece` maintain their advantage over earlier approaches for very large values of ϵ, around 30% of the entire range. Therefore, not only do our approaches provide improved space efficiency when high accuracy is required, but they also outperform earlier approaches when the error threshold is large. For values of ϵ that are larger than 30% of the dataset's range, only a handful of PLA segments are produced by Angle-based PLA, resulting in fewer opportunities for grouping similar segments by `Sim-Piece` and `Mix-Piece`. As a result, the Mixed and Slide algorithms perform better. However, the accuracy in such settings is clearly very low.

Table 4 Compression Ratio (CR), measured MAEr%, MAE and RMSE for $\epsilon = 5\%$ and $\epsilon = 0.5\%$

Epsilon		Dataset													
		Cricket 5%	FaceFour 5%	Lightning 5%	MoteStrain 5%	Wafer 5%	Wind speed 5%	Wind Dir. 5%	Pressure 0.5%	BTCUSD 0.5%	ETHUSD 0.5%	SPX 0.5%	STOXX50E 0.5%	Average/ % 5%	Dev. from Mix-Piece 0.5%
Mix-Piece	CR	**86.7**	**24.0**	**128.8**	**40.1**	**71.3**	**55.0**	**17.3**	**546.1**	**148.1**	**87.0**	**1,083.7**	**199.5**	**60.5**	**412.9**
	MAEr%	2.07%	2.27%	2.28%	2.33%	1.59%	2.40%	2.10%	0.22%	0.18%	0.19%	0.18%	0.19%	2.15%	0.19%
	MAE	0.475	0.240	0.568	0.401	0.242	0.488	7.565	0.029	45.919	1.257	2.934	3.421	1.426	10.712
	RMSE	0.565	0.284	0.658	0.473	0.317	0.581	9.140	0.034	55.494	1.516	3.532	4.129	1.717	12.941
Sim-Piece	CR	74.8	20.9	115.4	35.6	61.4	40.8	15.0	412.0	85.1	52.0	643.0	123.5	52.0/-14%	263.1/-36%
	MAEr%	2.21%	2.43%	2.26%	2.70%	2.75%	2.29%	2.46%	0.23%	0.20%	0.20%	0.19%	0.20%	2.44%/13%	0.20%/5%
	MAE	0.506	0.258	0.563	0.464	0.418	0.467	8.841	0.030	50.420	1.347	3.142	3.681	1.645/15%	11.724/9%
	RMSE	0.597	0.302	0.655	0.528	0.471	0.560	10.451	0.035	60.190	1.606	3.742	4.392	1.938/13%	13.993/8%
Mixed	CR	45.2	15.6	83.8	20.4	27.6	27.2	7.6	396.2	118.2	66.9	924.0	149.4	32.5/-46%	330.9/-20%
	MAEr%	2.32%	2.59%	2.47%	2.95%	2.86%	2.29%	2.64%	0.25%	0.20%	0.20%	0.19%	0.20%	2.59%/20%	0.21%/11%
	MAE	0.532	0.275	0.615	0.507	0.434	0.466	9.521	0.032	49.567	1.339	3.089	3.601	1.764/24%	11.526/8%
	RMSE	0.624	0.322	0.707	0.572	0.495	0.557	11.267	0.037	59.395	1.607	3.688	4.315	2.078/21%	13.808/7%
Slide	CR	38.3	11.9	67.9	17.5	26.1	24.3	6.7	312.2	100.4	57.2	783.2	127.9	27.5/-55%	276.2/-33%
	MAEr%	2.34%	2.65%	2.72%	2.89%	2.96%	2.41%	2.66%	0.24%	0.19%	0.20%	0.19%	0.20%	2.66%/24%	0.20%/5%
	MAE	0.535	0.281	0.677	0.497	0.449	0.491	9.576	0.031	49.465	1.334	3.108	3.649	1.787/25%	11.517/8%
	RMSE	0.626	0.327	0.765	0.562	0.507	0.583	11.320	0.036	59.268	1.602	3.706	4.366	2.099/22%	13.796/7%
Swing	CR	22.8	10.2	36.6	13.4	19.6	9.8	4.4	140.5	36.7	23.3	127.0	36.4	16.7/-72%	72.8/-82%
	MAEr%	2.47%	2.47%	2.44%	2.49%	2.02%	2.62%	2.29%	0.24%	0.24%	0.24%	0.23%	0.24%	2.40%/12%	0.24%/26%
	MAE	0.567	0.262	0.608	0.428	0.307	0.533	8.233	0.032	59.938	1.611	3.812	4.392	1.563/10%	13.957/30%
	RMSE	0.657	0.306	0.704	0.503	0.374	0.619	10.082	0.037	70.111	1.882	4.450	5.132	1.892/10%	16.322/26%

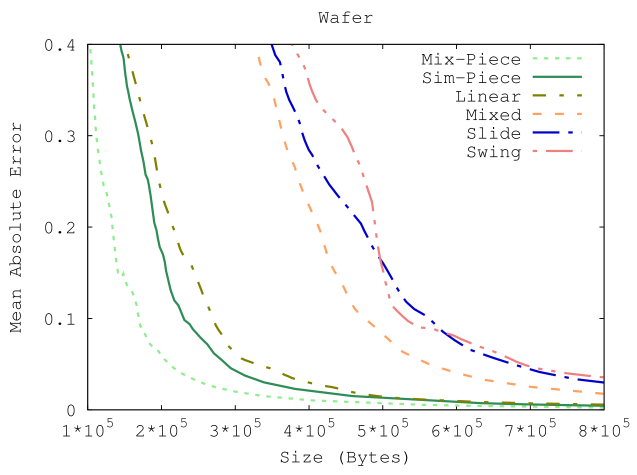

Fig. 10 The tradeoffs between quality, as assessed by Mean Absolute Error (MAE), and space efficiency provided by each PLA method. Notably, Mix-Piece outperforms all previously established techniques

4.5 Supplementary compression techniques

Our next experiment focuses on the use of supplementary compression mechanisms in conjunction with PLA, aiming to further reduce the space requirements. More specifically, we experimented with using delta encoding [39] on the timestamps of each PLA algorithm. Delta encoding results in smaller numeric values that could be stored using variable-byte encoding [41], reducing the size of their representation. Another novel technique for reducing the size of a PLA representation is to replace timestamps with indices that denote the length of a segment. This idea has been explored in [11] in the Single-Stream Protocol that uses 1-byte indices for representing the length of a segment. The protocol also permits compact representation of singleton values for segments with length < 3. Based on the evaluation in [11] and code that was kindly provided by the authors, we also present results for using this protocol with the Slide algorithm (denoted as Slide-SS). As a baseline we also provide results from passing the whole binary output of each PLA algorithm through the popular Zstandard [4] algorithm.

Figure 12 illustrates the space gains we obtain for all time series of our dataset. We used as ϵ the smallest value considered for each dataset in the results in Fig. 9. We see that Zstandard does help increase the compression ratio of all approaches, offering additional space savings. However, even with the benefit of using an additional general purpose compression algorithm, none of the earlier PLA approaches is able to compete against our newly proposed Mix-Piece algorithm, which clearly offers a better compression ratio *without* any added help. The savings produced from employing delta-encoding are notably substantial, primarily owing to the significant presence of timestamps within a PLA output. The reduction in the binary representation of these timestamps leads to gains surpassing those achieved by a

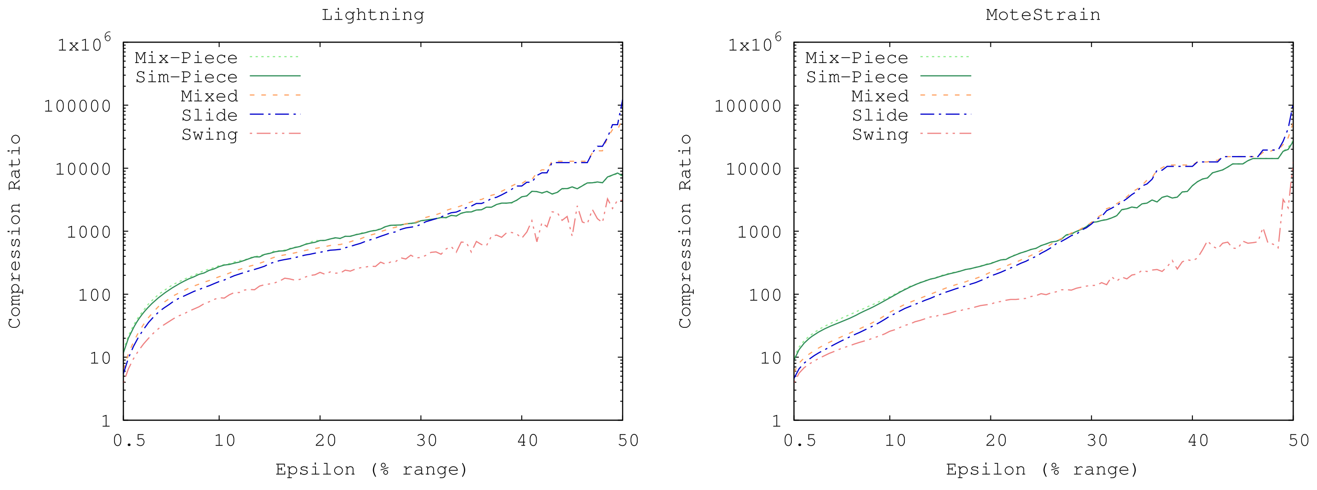

Fig. 11 Exploring larger error tolerances. `Mix-Piece` and `Sim-Piece` outperform earlier approaches up to very large values of ϵ

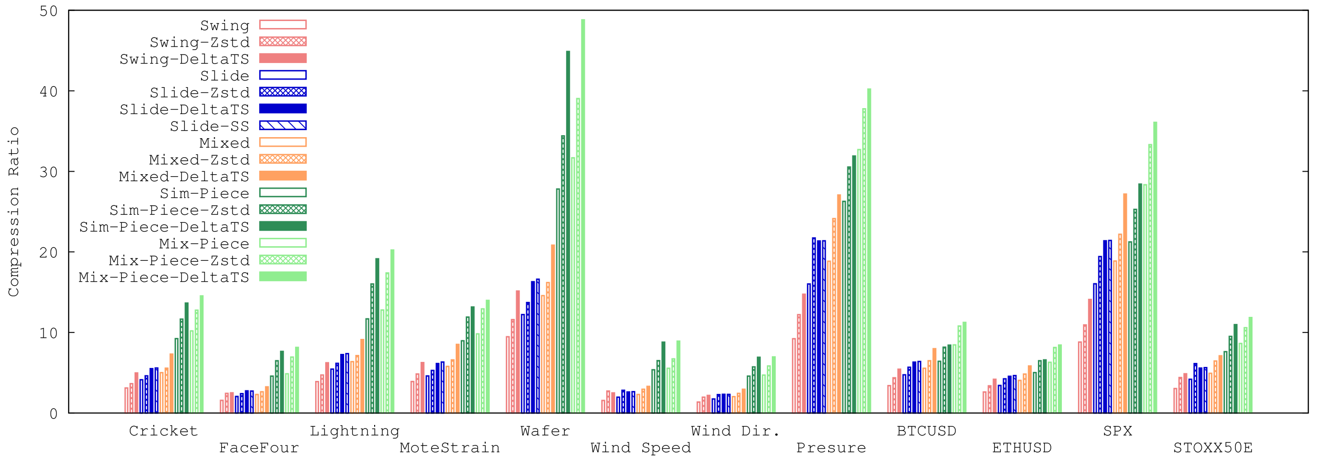

Fig. 12 Effect of applying supplementary compression to the output of PLA algorithms

generic compression algorithm. The Single-Stream protocol provides comparable results to delta encoding. Overall the combination of `Mix-Piece` with delta encoding of the timestamp values in its output provides the best compression gains.

4.6 Execution time

Table 5 reports the time required to process each of the time series datasets by the PLA algorithms at three distinct error thresholds (0.05%, 0.5%, and 5%). The reported results represent the average times calculated from 25 executions. Additionally, for `Sim-Piece` and `Mix-Piece` we present the total time required, along with a breakdown of the time spent on each of their two phases. Lastly, we calculate the average of execution time per PLA method and ϵ, and determine the percentage deviation of these averages from Mix-Piece.

From the data presented in Table 5, a notable observation is that `Mix-Piece` outperforms the second-best approach in terms of compression ratio, i.e., Mixed, by at least one order of magnitude in speed. When comparing `Sim-Piece` and `Mix-Piece` it becomes apparent that `Mix-Piece` does exhibit a slight decrease in speed. Specifically, during Phase 1 of `Mix-Piece`, the process of managing and updating two intervals results in a minor slowdown. However, Phase 2 shows slightly improved speed on average due to the reduced number of segments considered. Moreover, our results indicate that the performance of `Mix-Piece` is, in most cases, faster than the Slide algorithm as well.

The Swing algorithm, that provides the worst compression ratio among the approaches investigated here, is shown to be the fastest approach. However, the results of Table 5 clearly show that the execution time of `Mix-Piece` decreases as the value of ϵ grows, due to the smaller number of segments produced during Phase 1 of the algorithm. Thus, `Mix-Piece` becomes much more efficient as the value of

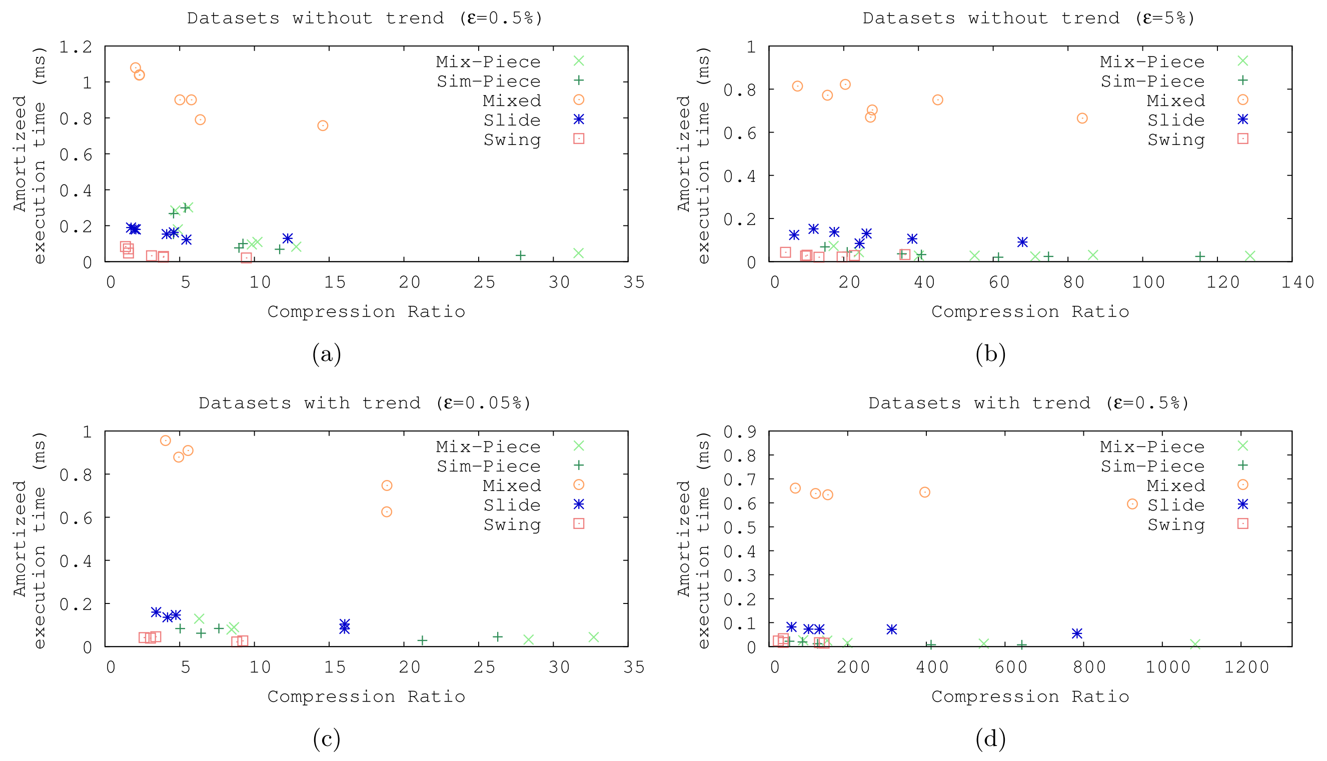

Fig. 13 Execution time (amortized per dataset record) vs compression ratio tradeoff

ϵ grows, providing significantly better compression ratio for all possible error-thresholds values (e.g., Fig. 11).

The overall superiority of `Mix-Piece` is evident in Fig. 13, that illustrates the trade-off between compression time and compression ratio achieved for all algorithms and datasets used in our experiments. To address the varying dataset lengths (as detailed in Table 1), we normalize the execution time of each algorithm by dividing it by the length of the corresponding dataset. In Figs. 13a and c, we use the smallest ϵ value considered for each dataset from the results in Fig. 9 for the datasets without trend (13a) and with trend (13c), respectively. Conversely, in Fig. 13b, d, we utilize the largest ϵ value. We clearly see how `Mix-Piece` provides impressive space savings while also being almost equivalently efficient with the fastest approach. The findings depicted in Fig. 13 establish our algorithm as the undisputed preferable option for space-efficient approximation of time series.

5 Related work

We review here existing approaches that are used to approximate time series data. We also discuss compression techniques studied in relevant fields, as well as two lossless compression algorithms.

An optimal algorithm for approximating sensor data using Piecewise Constant Approximation is given in [25]. A cache filter is used, which predicts that the next data point will have a value within the error threshold from the previous one. Thus, a new data point needs to be recorded only when it violates the error constraint. A similar approach is discussed in [34].

Elmeleegy et al. [12] propose Swing and Slide, a novel joint- and a disjoint-segment PLA algorithm, respectively. Similar algorithms had been independently discovered in earlier works, by Gritzali and Papakonstantinou [17] and O'Rourke [35]. Swing constructs the longest possible line segment starting from a fixed origin point by adjusting two bounding slope lines until reaching a break-up point, which will generate a new joint segment, starting from the last point of the previous segment. Slide filters are different as they generate disjoint segments as an approximation for the original data points. This gives them more flexibility at the expense of having to store an additional value for each segment. Updating the bounding slope lines can be optimized with the use of the convex hull of the observed data points, which allows for checking only against the points of the convex hull, instead of the significantly larger number of all the data points observed in the current filtering interval.

Swinging Door Trending (SDT) is a earlier algorithm similar to Swing [1]. It involves managing two sloping lines, upper and lower, each pivoting in opposite directions

Table 5 Execution time (in ms.) for 0.5% and 5% Epsilon

Epsilon		Dataset												
Mix-Piece		Cricket	FaceFour	Lightning	MoteStrain	Wafer	Wind speed	Wind Dir.	Pressure	BTCUSD	ETHUSD	SPX	STOXX50E	Average/ % Dev. from Mix-Piece
5%	Phase 1	14.16	1.03	2.94	1.69	16.71	50.06	26.69	79.30	1.84	1.75	39.96	11.63	20.65
	Phase 2	7.77	0.70	0.39	1.27	9.68	65.06	58.21	0.45	0.01	0.02	0.02	0.06	11.97
	Total	21.93	1.73	3.34	2.96	26.40	115.12	84.89	79.75	1.86	1.77	39.98	11.69	32.62
0.5%	Phase 1	21.08	2.87	4.97	4.75	29.63	242.89	60.41	141.50	2.37	2.38	17.73	14.92	45.46
	Phase 2	55.93	4.18	5.21	5.50	22.35	1,000.74	272.83	9.72	0.25	0.43	0.59	2.14	114.99
	Total	77.00	7.05	10.19	10.25	51.98	1,243.63	333.23	151.22	2.62	2.81	18.32	17.06	160.45
0.05%	Phase 1	39.66	2.25	6.23	4.13	24.83	282.63	93.60	192.96	3.75	5.99	30.33	30.73	59.76
	Phase 2	170.49	6.27	14.45	14.13	70.47	1,375.93	361.35	341.82	4.56	7.62	28.18	70.21	205.46
	Total	210.15	8.52	20.67	18.26	95.30	1,658.56	454.95	534.78	8.31	13.60	58.51	100.94	265.21
Sim-Piece														
5%	Phase 1	9.51	0.78	2.51	2.28	9.33	39.29	13.38	88.60	1.61	1.74	7.46	8.76	15.44/-25%
	Phase 2	7.51	0.97	0.49	1.60	13.21	96.69	66.97	0.39	0.00	0.01	0.01	0.04	15.66/31%
	Total	17.02	1.75	2.99	3.88	22.54	135.98	80.35	88.99	1.61	1.75	7.47	8.80	31.09/-5%
0.5%	Phase 1	9.88	1.49	2.33	1.80	10.12	117.32	32.76	82.44	1.86	2.01	12.85	11.16	23.83/-48%
	Phase 2	60.78	4.50	6.13	6.41	27.52	1116.66	280.22	11.78	0.18	0.36	0.49	2.34	126.45/10%
	Total	70.66	5.99	8.46	8.21	37.65	1233.98	312.98	94.22	2.05	2.37	13.34	13.50	150.28/-6%
0.05%	Phase 1	22.91	1.54	2.77	2.66	13.88	130.10	49.69	135.83	2.40	2.83	20.96	20.46	33.84/-43%
	Phase 2	165.80	5.59	14.62	13.39	75.18	1395.32	324.72	414.57	4.12	5.98	30.44	74.00	210.31/2%
	Total	188.71	7.13	17.39	16.05	89.06	1525.42	374.41	550.40	6.53	8.81	51.40	94.46	244.15/-8%
Mixed														
5%		526.80	30.25	81.62	87.83	766.11	2759.57	952.19	6750.19	63.45	70.63	976.49	658.26	1,143.62/3,406%
0.5%		632.16	40.76	96.93	96.20	824.48	4274.98	1261.89	7799.10	67.18	69.54	1,075.86	711.22	1,412.52/780%
0.05%		771.77	49.71	122.48	116.26	861.94	4536.06	1341.92	7561.32	95.63	100.50	1,350.47	986.13	1491.18/462%
Slide														
5%		74.62	5.96	11.17	14.74	142.33	348.48	144.91	657.67	4.59	4.89	68.17	48.04	127.13/290%
0.5%		107.55	7.03	15.04	17.40	141.07	746.75	222.07	874.75	7.68	8.64	99.20	81.32	194.04/21%
0.05%		138.56	7.92	21.18	20.37	138.18	796.18	235.74	994.31	15.41	16.82	189.22	152.77	227.22/-14%
Swing														
5%		19.94	1.18	4.04	2.18	23.63	108.25	50.65	175.87	3.48	3.37	29.73	18.38	36.73/13%
0.5%		22.99	1.88	3.05	3.09	22.54	293.48	98.48	176.72	3.50	2.51	29.92	19.48	56.47/-65%
0.05%		65.29	5.41	6.37	3.73	46.26	364.86	102.91	324.16	4.83	4.39	39.65	44.34	84.35/-68%

(counter-clockwise and clockwise) as it processes incoming data points. By utilizing these sloping lines, the algorithm calculates a parallelogram with a longitudinal trend line in the center. This helps delineate an area of input points that fall within the maximum allowable distance from the trend line. GreedyPLR [42] is a variant of Swing in which the point used to *swing* up and down is set to be the intersection of the upper and lower bounding slopes, instead of the starting point of a segment. This variation provides a wider angle than Swing while preserving a worst-case O(1) complexity. Similar to Slide [12] and the work in [35], OptimalPLR [42] uses convex hulls to find the optimal PLA with regard to the number of line segments using only disjoint segments and may offer faster processing time [10].

Hakimi and Schmeichel [20] solve optimally the continuous PLA problem, in which the approximation segments are forced to form a continuous function. This allows for representing the segments with two values, instead of the three required when using disjoint segments. The process is similar with that of OptimalPLR; however, each new segment starts from the previous generated line instead of the breakup point. The work in [20] is based on an algorithm proposed by Imai and Iri [21]. A variation of the work in [20] is given in [11], where a simple regression model is used to obtain the best-fit line at the cost of increased complexity.

Luo et al. [29] introduce the problem of *mixed-type* PLA, aiming to optimize the representation size through an adaptive solution that uses a mixture of joint and disjoint segments. Their novel approach allows for enjoying the *best of two worlds*, i.e., the cheaper representation of joint segments and the fewer number of segments produced by disjoint-segment approaches. A dynamic programming algorithm is presented that finds the optimal sized PLA in under these settings. Moreover, the authors avoid wasting a bit for each segment to differentiate between joint and disjoint segments. Instead, the proposed representation exploits the strictly increasing sequence of positive timestamps and uses negative values to indicate a disjoint segment.

Compressing time series data in streaming applications is a computation step that inherently introduces latency. Most algorithms do not take into consideration, for example, how often should the calculated compressed representation be output [10, 11]. PLA techniques face this issue as they inherently try to group multiple incoming observations into a segment as large as possible. Similarly, at the receiving endpoint, running the reconstruction phase, the receiver often may need to wait for the next segment to reconstruct the previous one. Notably, the work in [10, 11] proposes streaming protocols as algorithmic implementations of several state of the art PLA techniques (joint/disjoint or mixed). Our `Mix-Piece` algorithm as a two-step process, introduces additional latency in the output, as the final grouping of the segments necessitates the completion of the sorting phase. Nevertheless, for applications that require low-latency access to the compressed time series values, one may tap into the output of the first phase of the algorithm, that is essentially a standard disjoint PLA representation and utilize techniques such as those discussed in [10, 11] to reduce the perceived latency.

Compression of time series has been studied in the context of other work as well. The authors of [7, 8] proposed a technique that exploits the correlation and redundancy among multivariate sensor readings to construct a dictionary of real measurements, used for encoding piece-wise linear correlations among the collected data values. [9, 14] extend these techniques to work in a distributed setting of network-connected nodes, while also accounting for the presence of outliers. In [44], the authors propose an approach for efficiently processing time series *similarity* matching queries, by dividing time series into a fixed number of equal sized segments. Guerts [13] focuses on classification of time series and uses regression trees to perform piecewise constant modeling of temporal signals. Patterns are extracted from these models and are combined in decision trees to give interpretable classification rules.

In the field of lossless time series compression, Gorilla [37] is a compression algorithm, particularly suited for floating-point time series in which the neighboring data points do not change significantly. Timestamps and values are compressed separately, similarly to other time-aware schemes [27]. BUFF [28] uses a byte-oriented columnar storage representation for compressing bounded, low-precision floating values. A recent lossy compression technique, MOST [43], has been shown to improve compression by identifying outlier values in the time series, setting them aside and optimizing their encoding via a novel quantization technique. This is orthogonal to our work as the core idea of `Sim-Piece` and `Mix-Piece` is to group similar PLA segments. For lossless compression, the CHIMP [26] algorithm has been shown to be very competitive with *lossy* PLA approaches in terms of space requirements. Our algorithms are motivated by the impressively low space requirements of CHIMP and achieve significant improvements with regards to the state-of-the-art in PLA algorithms, offering better compression than CHIMP, even when the precision error threshold is set extremely low.

Compression of time series data exhibits similarities with the problem of constructing histograms for approximating frequency distributions. For a given space constraint both problems can be solved optimally employing a straightforward but time-consuming dynamic programming algorithm, as highlighted in [22]. Motivated by wavelets, other techniques exploit a hierarchical decomposition of the data space [23, 30] to increase the accuracy of the histogram synopsis. Some approaches also consider data uncertainty [5, 6], as is often the case with sensors readings that produce time series data. Although constructing histograms is typically treated as an offline task, there are methods available to adapt to rapidly changing data [15, 18].

6 Conclusions

In this paper, we have introduced innovative approaches for identifying similarities among line segments generated by a Piecewise Linear Approximation scheme. These approaches aim to create a unified, efficient representation for multiple segments. Our proposed algorithm, referred to as `Mix-Piece` builds upon a previously published method (`Sim-Piece`) in novel ways, yielding substantial and consistent improvements. These improvements were validated across a variety of datasets with differing characteristics.

Our flexible approach to grouping and representing line segments within the output of `Mix-Piece` allows us to achieve compression ratios that surpass those of existing state-of-the-art PLA algorithms. Consequently, `Mix-Piece` produces PLA approximations that enable the compact representation of large time series with high accuracy and improved efficiency in terms of both storage space and compression speed.

Acknowledgements This work has received funding from the Hellenic Foundation for Research and Innovation (HFRI) and the General Secretariat for Research and Technology (GSRT), under Grant Agreement No. 779.

References

1. Bristol, E.H.: Swinging door trending: adaptive trend recording? In: ISA National Conference Proceedings, pp. 749–753 (1990)

2. Cameron, S.H.: Piece-wise linear approximations. Technical report, IIT Research Inst Chicago, IL, Computer Sciences Div (1966)
3. Chen, Y., Keogh, E., Hu, B., Begum, N., Bagnall, A., Mueen, A., Batista, G.: The ucr time series classification archive (2015). www.cs.ucr.edu/~eamonn/time_series_data/
4. Collet, Y.: Zstd (2015). https://facebook.github.io/zstd
5. Cormode, G., Deligiannakis, A., Garofalakis, M.N., McGregor, A.: Probabilistic histograms for probabilistic data. Proc. VLDB Endow. **2**(1), 526–537 (2009). https://doi.org/10.14778/1687627.1687687, http://www.vldb.org/pvldb/vol2/vldb09-394.pdf
6. Cormode, G., Garofalakis, M.N.: Histograms and wavelets on probabilistic data. IEEE Trans. Knowl. Data Eng. **22**(8), 1142–1157 (2010). https://doi.org/10.1109/TKDE.2010.66
7. Deligiannakis, A., Kotidis, Y.: Data reduction techniques in sensor networks. IEEE Data Eng. Bull. **28**(1), 19–25 (2005). http://sites.computer.org/debull/A05mar/kotidis.ps
8. Deligiannakis, A., Kotidis, Y., Roussopoulos, N.: Compressing historical information in sensor networks. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13–18, 2004, pp. 527–538. ACM (2004). https://doi.org/10.1145/1007568.1007628
9. Deligiannakis, A., Kotidis, Y., Roussopoulos, N.: Dissemination of compressed historical information in sensor networks. VLDB J. **16**(4), 439–461 (2007). https://doi.org/10.1007/s00778-005-0173-5
10. Duvignau, R., Gulisano, V., Papatriantafilou, M., Savic, V.: Piece-wise linear approximation in data streaming: algorithmic implementations and experimental analysis. CoRR **abs/1808.08877** (2018)
11. Duvignau, R., Gulisano, V., Papatriantafilou, M., Savic, V.: Streaming piecewise linear approximation for efficient data management in edge computing. In: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, April 8–12, 2019, pp. 593–596. ACM (2019). https://doi.org/10.1145/3297280.3297552
12. Elmeleegy, H., Elmagarmid, A.K., Cecchet, E., Aref, W.G., Zwaenepoel, W.: Online piece-wise linear approximation of numerical streams with precision guarantees. Proc. VLDB Endow. **2**(1), 145–156 (2009). https://doi.org/10.14778/1687627.1687645, http://www.vldb.org/pvldb/vol2/vldb09-573.pdf
13. Geurts, P.: Pattern extraction for time series classification. In: Principles of Data Mining and Knowledge Discovery, pp. 115–127 (2001)
14. Giatrakos, N., Kotidis, Y., Deligiannakis, A., Vassalos, V., Theodoridis, Y.: TACO: tunable approximate computation of outliers in wireless sensor networks. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6–10, 2010, pp. 279–290. ACM (2010). https://doi.org/10.1145/1807167.1807199
15. Gilbert, A.C., Guha, S., Indyk, P., Kotidis, Y., Muthukrishnan, S., Strauss, M.: Fast, small-space algorithms for approximate histogram maintenance. In: J.H. Reif (ed.) Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19–21, 2002, Montréal, Québec, Canada, pp. 389–398. ACM (2002). https://doi.org/10.1145/509907.509966
16. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs/Martin Charles Golumbic. Academic Press, New York (1980)
17. Gritzali, F., Papakonstantinou, G.: A fast piecewise linear approximation algorithm. Signal Process. **5**(3), 221–227 (1983). https://doi.org/10.1016/0165-1684(83)90070-1
18. Guha, S., Koudas, N., Shim, K.: Approximation and streaming algorithms for histogram construction problems. ACM Trans. Database Syst. **31**(1), 396–438 (2006). https://doi.org/10.1145/1132863.1132873
19. Gupta, U.I., Lee, D.T., Leung, J.Y.: Efficient algorithms for interval graphs and circular-arc graphs. Networks **12**(4), 459–467 (1982). https://doi.org/10.1002/net.3230120410
20. Hakimi, S.L., Schmeichel, E.F.: Fitting polygonal functions to a set of points in the plane. CVGIP Graph. Model. Image Process. **53**(2), 132–136 (1991). https://doi.org/10.1016/1049-9652(91)90056-P
21. Imai, H., Iri, M.: An optimal algorithm for approximating a piecewise linear function. J. Inf. Process. **9**(3), 159–162 (1986)
22. Jagadish, H.V., Koudas, N., Muthukrishnan, S., Poosala, V., Sevcik, K.C., Suel, T.: Optimal histograms with quality guarantees. In: Gupta, A., Shmueli, O., Widom, J. (eds) VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24–27, 1998, New York, pp. 275–286. Morgan Kaufmann (1998). http://www.vldb.org/conf/1998/p275.pdf
23. Karras, P., Mamoulis, N.: Lattice histograms: a resilient synopsis structure. In: Alonso, G., Blakeley, J.A., Chen, A.L.P. (eds) Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7–12, 2008, Cancún, Mexico, pp. 247–256. IEEE Computer Society (2008). https://doi.org/10.1109/ICDE.2008.4497433
24. Kitsios, X., Liakos, P., Papakonstantinopoulou, K., Kotidis, Y.: Sim-piece: Highly accurate piecewise linear approximation through similar segment merging. Proc. VLDB Endow. **16**(8), 1910–1922 (2023). https://www.vldb.org/pvldb/vol16/p1910-liakos.pdf
25. Lazaridis, I., Mehrotra, S.: Capturing sensor-generated time series with quality guarantees. In: Proceedings of the 19th International Conference on Data Engineering, March 5–8, 2003, Bangalore, India, pp. 429–440. IEEE Computer Society (2003). https://doi.org/10.1109/ICDE.2003.1260811
26. Liakos, P., Papakonstantinopoulou, K., Kotidis, Y.: Chimp: efficient lossless floating point compression for time series databases. Proc. VLDB Endow. **15**(11), 3058–3070 (2022)
27. Liakos, P., Papakonstantinopoulou, K., Stefou, T., Delis, A.: On compressing temporal graphs. In: 38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9–12, 2022, pp. 1301–1313. IEEE (2022). https://doi.org/10.1109/ICDE53745.2022.00102
28. Liu, C., Jiang, H., Paparrizos, J., Elmore, A.J.: Decomposed bounded floats for fast compression and queries. Proc. VLDB Endow. **14**(11), 2586–2598 (2021). https://doi.org/10.14778/3476249.3476305, http://www.vldb.org/pvldb/vol14/p2586-liu.pdf
29. Luo, G., Yi, K., Cheng, S., Li, Z., Fan, W., He, C., Mu, Y.: Piecewise linear approximation of streaming time series data with max-error guarantees. In: 31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13–17, 2015, pp. 173–184. IEEE Computer Society (2015). https://doi.org/10.1109/ICDE.2015.7113282
30. Matias, Y., Vitter, J.S., Wang, M.: Wavelet-based histograms for selectivity estimation. In: Haas, L.M., Tiwary, A. (eds) SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2–4, 1998, Seattle, Washington, USA, pp. 448–459. ACM Press (1998). https://doi.org/10.1145/276304.276344
31. National Ecological Observatory Network (NEON): Barometric pressure above water on-buoy (dp1.20004.001) (2022). https://doi.org/10.48443/V4AP-NY05, https://data.neonscience.org/data-products/DP1.20004.001/RELEASE-2022
32. National Ecological Observatory Network (NEON): Windspeed and direction above water on-buoy (dp1.20059.001) (2022). https://doi.org/10.48443/E16C-8686, https://data.neonscience.org/data-products/DP1.20059.001/RELEASE-2022
33. Nicoomanesh, A.: Huge stock price data: Intraday minute bar (2021). https://www.kaggle.com/datasets/arashnic/stock-data-intraday-minute-bar

34. Olston, C., Jiang, J., Widom, J.: Adaptive filters for continuous queries over distributed data streams. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9–12, 2003, pp. 563–574. ACM (2003). https://doi.org/10.1145/872757.872825
35. O'Rourke, J.: An on-line algorithm for fitting straight lines between data ranges. Commun. ACM **24**(9), 574–578 (1981). https://doi.org/10.1145/358746.358758
36. Pavlidis, T.: Waveform segmentation through functional approximation. IEEE Trans. Comput. **22**(7), 689–697 (1973). https://doi.org/10.1109/TC.1973.5009136
37. Pelkonen, T., Franklin, S., Cavallaro, P., Huang, Q., Meza, J., Teller, J., Veeraraghavan, K.: Gorilla: A fast, scalable, in-memory time series database. Proc. VLDB Endow. **8**(12), 1816–1827 (2015). https://doi.org/10.14778/2824032.2824078, http://www.vldb.org/pvldb/vol8/p1816-teller.pdf
38. Stone, H.: Approximation of curves by line segments. Math. Comput. **15**(73), 40–47 (1961)
39. Suel, T.: Delta Compression Techniques. In: Sakr, S., Zomaya, A.Y. (eds) Encyclopedia of Big Data Technologies. Springer (2019). https://doi.org/10.1007/978-3-319-63962-8_63-1
40. Wang, H., Song, S.: Frequency domain data encoding in apache iotdb. Proc. VLDB Endow. **16**(2), 282–290 (2022). https://www.vldb.org/pvldb/vol16/p282-song.pdf
41. Williams, H.E., Zobel, J.: Compressing integers for fast file access. Comput. J. **42**(3), 193–201 (1999). https://doi.org/10.1093/comjnl/42.3.193
42. Xie, Q., Pang, C., Zhou, X., Zhang, X., Deng, K.: Maximum error-bounded piecewise linear representation for online stream approximation. VLDB J. **23**(6), 915–937 (2014). https://doi.org/10.1007/s00778-014-0355-0
43. Yang, Z., Chen, S.: MOST: model-based compression with outlier storage for time series data. Proc. ACM Manag. Data **1**(4), 250:1-250:29 (2023). https://doi.org/10.1145/3626737
44. Yi, B., Faloutsos, C.: Fast time sequence indexing for arbitrary lp norms. In: VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10–14, 2000, Cairo, Egypt, pp. 385–394. Morgan Kaufmann (2000)